NPS52-83-006

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DESIGN AND ANALYSIS OF A MULTI-BACKEND DATABASE SYSTEM FOR PERFORMANCE IMPROVEMENT, FUNCTIONALITY EXPANSION AND CAPACITY GROWTH (PART I)

David K. Hsiao and M. Jaishankar Menon

June 1983

Prepared for: Naval Postgraduate School
Monterey, CA 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral J. J. Ekelund
Superintendent

David A. Schrady
Provost

The work reported herein was supported by Contract N00014-75-C-0573 from the Office of Naval Research.

This report was prepared by:

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-83-006	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) David K. Hsiao and M. Jaishankar Menon		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Chief of Naval Research Arlington, VA 22217		12. REPORT DATE June 1983 13. NUMBER OF PAGES 175
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Multi-backend systems, taxonomy of database systems, channel limitation problem, Device limitation problem, software and hardware specialization problems, performance enhancement, capacity growth, closed network queueing model, request execution, directory management strategies, data placement strategies.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

It is generally known that the use of a single general-purpose digital computer with dedicated software for database management as a backend to offload the mainframe host computer from database management tasks yields no appreciable gains in performance and functionality. Research is therefore being pursued to replace this software backend approach to database management with an architecture approach which will yield good performance and new functionality.

The aim of the proposed research is to investigate whether for the management of very large databases the use of multiple mini-computer systems in a parallel

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

configuration is feasible and desirable. By feasible we mean that it is possible to configure a number of (slave) minicomputers each of which is driven by identical database management software and controlled by a (master) minicomputer for concurrent operations on the database spread over the disk storage local to the slave computers. This approach to large databases may be desirable because only off-the-shelf equipment of the same kind is utilized to achieve high performance without requiring specially-built hardware and because identical database management software is replicated on the slave computers. The approach makes the capacity growth and performance improvement easy because duplicate hardware can be added and used with replicable software.

To study the feasibility, we research into the software architecture issues and the hardware limitations of the master and slaves. We also research into the replicable software for the slaves. Since these slaves are to be operated concurrently with corresponding single-channel disks, we can investigate the effects of either single-query-and-multiple-database-streams or multiple-queries-and-multiple-database-streams operations for performance improvement. To study the desirability, we intend to consider the factors related to the problem of capacity growth and cost effectiveness. The central issue may be whether we can realize a high-performance and great-Capacity database management system with a multiplicity of minicomputers, large number of single-channel disks and replicable software cost-effectively.

In this report, we present a new approach to the solution of database management problems involving database growth and performance enhancement. A system which uses a multiplicity of conventional minicomputers, novel hardware configuration and innovative software design is presented. This extensible system tries to achieve the ideal goal of having the performance (both response time and throughput) be proportional to the multiplicity of minicomputers.

Our first effort is to identify the problems and bottlenecks involved in developing such an ideal system. Two major problems, one called the controller limitation problem and the other the channel limitation problem are identified. Having identified these problems, our next effort is to systematically eliminate or minimize the ill-effects of these problems. We have also identified a number of other problems.

For studying the multiple back-end database system, we utilize queueing models and simulation. Queueing models and simulation are used at different design stages in order to aid the design process. Finally, ours is the only comprehensive design of a multiple backend system that covers all aspects of database management. Algorithms for the four basic request types (insert, retrieve, delete and update) algorithms for aggregate operations, algorithms for access control, algorithms for concurrency control, algorithms for database reorganization and algorithms for addition of new backends are all analyzed and specified.

Because of its volume, the design and analysis is presented in two parts. In the Part I, we include four chapters. In Chapter I, we set our aim and scope of the research. In Chapter II, we will make a survey of typical software-oriented multiple backend systems in existence. We will point out the strengths and weakness of these systems and make some recommendations so that the weaknesses pointed out for some of the systems can be avoided. One of the recommendations is on data placement strategies. As a multiple-data-stream computer, the multi-backend database system must be able to move data across the backends parallelly. Thus, the placement of the database over separate disk systems for the corresponding backends constitutes an important issue for study. We have proposed and analyzed a number of strategies for data placement. We also select one for our system. In Chapter III, we will select a data model and data manipulation language for implementation. We argue that the attribute-based model is the "superior" or "most appropriate" data model. Our main argument is that prevailing data models and languages such as relational, network and hierarchical can be converted and translated into the attribute-based model in a straightforward manner, thus, allowing the system to support a number of models and languages. Accordingly, a simple data mani-

ABSTRACT continued

pulation language based on this attribute-based model is chosen and presented. In Chapter IV, we elaborate on the process of request execution. These are the requests for record retrieval, insertion, deletion and update. Central to the process of request execution is the study of information about the database. We propose a new method for record clustering and a new strategy for directory management. In particular, the directory response time, throughput and storage requirements are analyzed. A closed queueing network model which incorporates a separate I/O submodel for the disk subsystem is used for this study.

The rest of the design and analysis will be included in Part II of this report. The entire design and analysis are based on analytical and simulation studies. An implementation effort will be on its way. It is hoped that we can test the system experimentally soon.

PREFACE

This work was supported by Contract N00014-75-0573 from the Office of Naval Research to Dr. David K. Hsiao and conducted in the Laboratory for Database Systems Research. The Laboratory for Database Systems Research is initially funded by the Digital Equipment Corporation (DEC), Office of Naval Research (ONR) and the Ohio State University (OSU) and consists of the staff, graduate students, undergraduate students, visiting scholars and faculty for conducting research in database systems. Dr. Douglas S. Kerr, Associate Professor of Computer and Information Science at the Ohio State University, is the present Director of the Laboratory.

Since July 1, 1982, Dr. Hsiao assumed the Chairmanship of the Computer Science Department at the Naval Postgraduate School and continued the funded research at the Naval Postgraduate School. The Laboratory for Database Systems Research is being moved to the Naval Postgraduate School (NPS) in June of 1983 and supported by DEC, ONR, and NPS. This report is a re-issue of an earlier report published in July 1981 at the Ohio State University under report number OSU-CISRC-TR-81-7.

We would like to thank all those who have helped with the MDBS project. In particular, the MDBS design and analysis were developed by Jai Menon. (Now, Dr. Jai Menon of IBM Research Laboratory, San Jose, California.) He also provided much help in the detailed designs. A visiting scholar, Xing-Gui He, is involved with MDBS project. Several undergraduate students are also involved with the project: Raymond Browder, Chris Jeschke, Jim McKenna, and Joe Stuber. Several graduate students, visiting scholars and undergraduate students provided much help in the detailed design and coding: Steven Barth, Julie Bendig, Abdulrahim Beram, Patti Dock, Masanobu Higashida, Jim Kiper, Drew Logan, William Mielke, Tamer Ozsu, Zong-Zhi Shi, and Paula Strawser. Jose Alegria, Tom Bodnovich and David Brown contributed background material which was necessary for making our design decisions. We would also like to thank the laboratory staff and other operators who provided us with system support: Bill Donovan, Doug Karl, Paul Placeway, Steve Romig, Jim Skon, Dennis Slaggy, Mark Verber, and Geoff Wyant.

TABLE OF CONTENTS

TABLE OF CONTENTS continued

LIST OF FIGURES

LIST OF FIGURES continued

TABLE OF CONTENTS continued

TABLE OF CONTENTS continued

TABLE OF CONTENTS continued

1. INTRODUCTION

For solving database management problems, many researchers [Babb79, Bane78, Cope73, Schu79, Wah80] have proposed solutions utilizing special-purpose hardware, known as database machines. However, it has not yet been demonstrated that these database machines are cost-effective. A different approach to the solution of database management problems may involve the use of conventional hardware elements, perhaps in large number, with most of the database management functions carried out in software. In other words, we may be searching for hardware solutions without having considered all possible software solutions to database management problems. In this dissertation, we emphasize the software approach to the solution of database management problems.

1.1 The Goal

The goal is to investigate whether it is possible to use a multiplicity of general-purpose processing and storage elements (specifically, minicomputers and disk drives), novel hardware configuration and innovative software design for achieving throughput gain and response-time improvement over conventional database management systems. Ideally, the performance gains and improvements should be proportional to the multiplicity of processing and storage elements used.

If this ideal goal cannot be achieved, then the case for database machines that use special-purpose hardware becomes very strong. On the other hand, if it can be shown that such an ideal system can be obtained, then a cost-effective way for high-volume and great-capacity database management may become more readily available. Database machines may still provide better performance, but they will be more expensive. Furthermore, database machines may represent a distant solution whose time is not yet near.

In this dissertation, we will propose a hardware configuration of multiple minicomputer systems and a design of a software database management system for the configuration. We will use analytical techniques and simulation studies to determine how closely the ideal goal has been achieved by our proposed configuration and design.

1.2 A Taxonomy of Existing Systems

One way to give some perspective to our work is by way of a taxonomy of database systems and machines. By developing the taxonomy and indicating the relative position of our work within this taxonomy, we may also explain the

similarities and differences of our work with that of others which are either operational or being proposed. Finally, we will show the advantages of the software approach over the machine approach. The taxonomy we developed is shown in Figure 1.

At the highest level, we differentiate between systems which utilize a central controller to simplify the control functions and those which do not utilize a central controller. In general, it is the case that systems which do not utilize a central controller are those where the database is geographically dispersed and the various computers have to be connected together by a network. Examples of such systems are SDD-1 [Roth80], Distributed Ingres [Ston76a], and Muffin [Ston79]. Also, generally speaking, systems that do not use a central controller will have either partially or fully duplicated databases. The need for having duplicate databases becomes important in a geographically dispersed database where data transfers among computers via a network are expensive. Hence, it is important to duplicate data so that the data is very close to the point where it is actually needed. In systems with duplicate databases, the concurrency control algorithms are mostly complex and require large numbers of messages to be exchanged [Hsia81]. This dissertation is not concerned with systems that do not utilize a central controller and where the databases are duplicated, though this could very well be an area for future research.

In developing our taxonomy further, we divide the systems with central controllers into two classes, namely, the hardware-oriented systems and the software-oriented systems. Hardware-oriented systems (also called database machines) are those which typically use special-purpose hardware to perform a large number of the database management functions. These systems [Bane78, Cope73, Schu79] process the data 'on the fly' while the data is being read from the disk tracks. Special-purpose processors which are associated with blocks of secondary storage are utilized to perform on-the-fly processing. Our taxonomy is, of course, subjective in defining a hardware-oriented system in terms of the number of the database management functions being accomplished in hardware. What is a 'large' number of the database management functions? Clearly, there will be some systems which fall in the borderline between hardware-oriented and software-oriented systems. Note that we do not consider a database management system such as DIRECT [Dewi78] to be a hardware-oriented system even though it is often referred to as a database machine in the literature. This is because the only special-

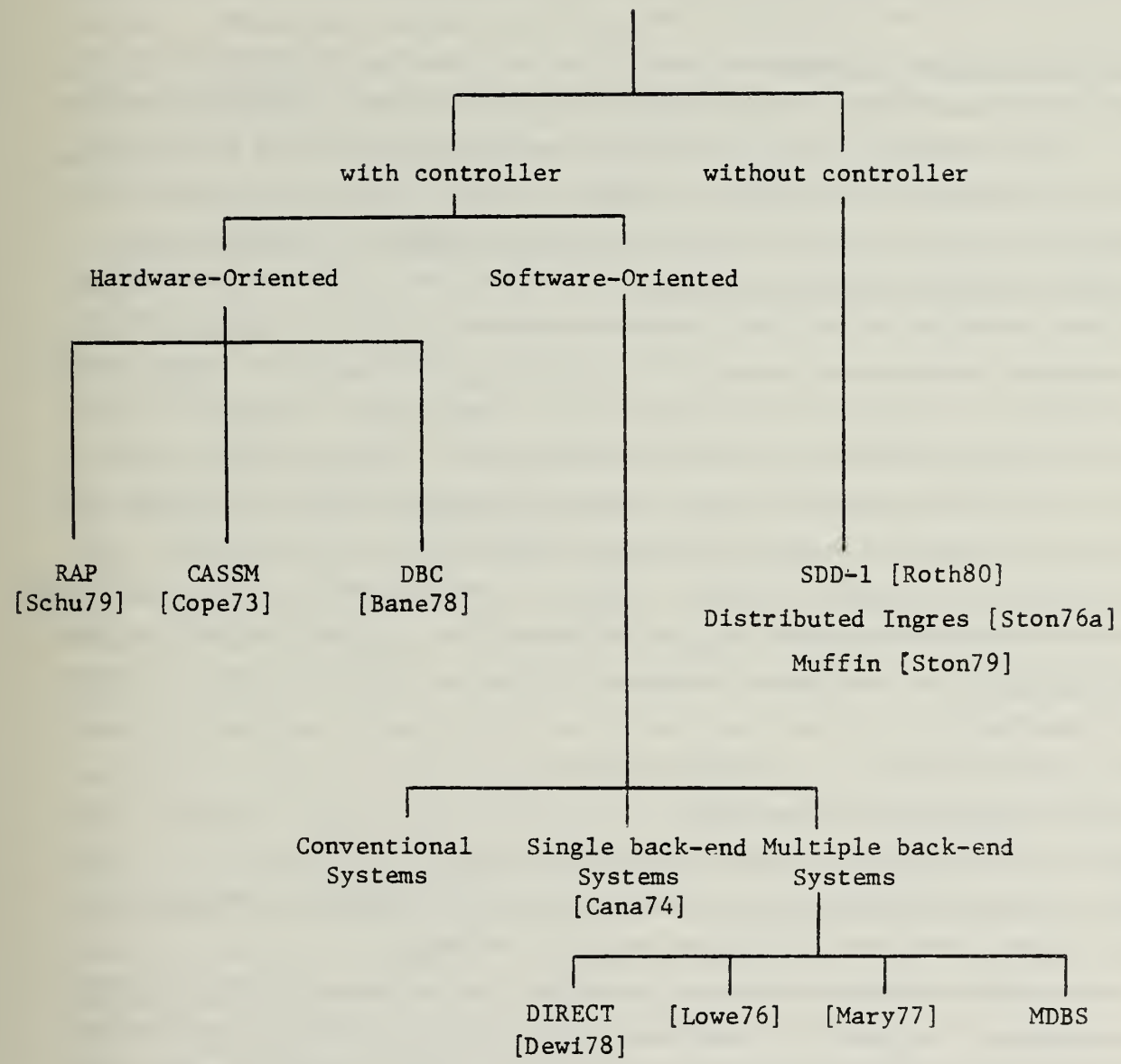


Figure 1. A Taxonomy of Database Management Systems

purpose hardware in DIRECT is a crossbar switch. All the database management functions in DIRECT are performed by software in conventional processors. Hence, by our classification, DIRECT is a software-oriented database management system.

Software-oriented systems are those which do not use a significant amount of special-purpose hardware and where most of the functions of database management are done in software. Such systems may be further subdivided into three categories as conventional database management systems, single back-end database management systems and multiple back-end database management systems. In a conventional database management system, all of the major software components, such as the operating system, the database management system and user (application) programs, are executed in a single computer system which has direct access to the database stored in the secondary memory. Performance upgrades in a conventional database management system are usually costly and disruptive. For instance, it may be necessary to add large memory modules or to incorporate more channels or to replace the central processor with a more powerful model. Such upgrades often require major effort so that it is well accepted that conventional database management systems are not easily extensible. In this context, we define extensibility of a database management system as the capability of the system for upgrade with

(1) no modification to existing software,

(2) no additional programming,

(3) no modification to existing hardware and

(4) no major disruption of system activity when additional hardware is being incorporated into the existing hardware.

A new configuration for database management called the back-end configuration was proposed in [Cana74]. Excellent descriptions of the back-end concept and its advantages and disadvantages are given in [Lowe76, Mary80] and will not be repeated here. Briefly, this configuration utilizes two computer systems -- a host and a back-end. The database management functions are implemented on the back-end which has exclusive access to the database on secondary storage devices. We shall refer to such a system as a single back-end database management system, since multiple back-end systems have also been proposed [Lowe76, Ston78, Dewi78]. Among the advantages claimed of single back-end database management systems is that performance upgrades are less disruptive, more manageable and on a smaller scale than in a conventional database management system.

First, upgrading the back-end requires no modification to application programs since they are executed in the host. Furthermore, the back-end separates the characteristics of the secondary storage devices from the characteristics of the host. Thus, new storage technology may be employed without changing hardware or software in the host. However, single back-ends have the disadvantage that, ultimately, performance upgrades will require replacement of the back-end and this may entail software modifications and hardware disruption.

The next logical step in the evolution of database management systems is the multiple back-end system approach as suggested by a number of researchers [Bane78, Dewi78, Lowe76, Ston78]. This approach employs multiple computer systems for database management with a software-oriented system and a centralized controller. A few words regarding the differences between our system and other software-oriented multiple back-end systems with controllers [Lowe76, Ston78, Dewi78] is now in order. The work of [Lowe76] did not represent the design of any specific database management system. That work was essentially in the nature of an idea for an architecture which others could utilize to design systems. The works of [Dewi78, Ston78], however, are actual designs. The difference between our work and that of these two researchers are explained at a very detailed level in Chapter2. Here, we satisfy ourselves with some differences in terms of overall objectives. None of the above-mentioned researchers emphasize the extensibility of the multiple back-end approach nor do they seem very concerned about designing their respective multiple back-end systems to make them extensible. In [Lowe76], for example, the multiple back-end system is considered from the viewpoint of enhanced reliability over single back-end systems. In [Dewi78, Ston78], however, the emphasis is on the general notion that multiple back-ends may provide greater throughput than single back-end systems. However, we believe in more specific findings. In other words, we believe that there are designs which can achieve the ideal goal of the throughput improvement being proportional to the multiplicity of back-ends. None of the above-mentioned systems has revealed such a design. We, on the other hand, reveal a design of a multiple back-end system which could lead to a response time being inversely proportional to the multiplicity of back-ends. It is also our belief that our multiple back-end system is extensible. That is, new storage devices and back-ends may be added to the configuration to improve its storage and performance capabilities without the need of re-designing and re-programming of the software and without major disruption of the existing hardware. We will be inter-

ested in multiple back-ends from the viewpoint of extensibility to improve response time and throughput. Furthermore, we will not utilize any special-purpose hardware in our system.

As a result of the extensible nature of our system, we make the following claim about our system. Consider a user of our database management system who finds that the system is saturated due to database growth and transaction increase. The user would like to upgrade the present system. The designers of database machines and conventional database management systems would offer the user the alternative of using their database machines and their systems, respectively, thereby replacing the existing software and hardware. Instead, we offer the user a different alternative. We do not ask the user to replace the existing software, since existing software is extensible and requires only a system generation. We also do not ask the user to replace the existing hardware. We merely ask the user to add identical hardware; consequently, when and if the need to upgrade the system arises, the user simply adds more hardware and uses the same existing software for performance gain and capacity growth.

The presentation of the taxonomy and the place of our system in that taxonomy has made clear some of the advantages and unique characteristics of the software approach to database management system design which we intend to follow. Two other considerations confirm that this may be a viable alternative to follow. We may term these as hardware considerations and software considerations. Both of these are considered in terms of our software approach and the database machines approach. First of all, from a hardware point of view, the conventional processors and disk drives will be cheaper than special-purpose processors and disk drives which may be needed in a database machine. From a software point of view, software for a large number of processors may be needed in this approach. However, novel software design may be used to ensure that the software in each of the minicomputers is identical. Thus, the software complexity is <u>not</u> proportional to the multiplicity of minicomputers. Furthermore, we are trying to support large databases that evolve and grow over time. This growth process is generally gradual. However, performance upgrades are necessary from time to time for such database growth. We have already indicated that our particular approach to database management systems can lead to extensible systems and improved performance to accomodate the growth and the evolution. Thus, from a software, hardware and database growth point of view, the software approach

using multiple back-ends seems preferable to the database machines approach. Thus, such an approach is certainly worth investigating and that is the focus of this dissertation.

1.3 Design Issues To Be Studied

Before an extensible system can be developed, a number of issues related to multiple back-end systems must be resolved. None of the researchers mentioned above have provided solutions to all these issues, though some researchers have proposed solutions to a subset of them. Let us enumerate below the various issues to be considered in the design of a multiple back-end system. The issues may be divided into three broad categories as hardware issues, system issues and software issues.

1.3.1 Hardware Issues

A. The Back-end Interconnection

The questions to be resolved here are, "What is the optimal way of interconnecting the back-ends together?", and "What is the optimal way of connecting the host to the back-ends?" In answering these questions, we must recall that the throughput of multiple processor systems increases significantly only for the first few additional processors. At some point, the throughput actually begins to decrease with each additional processor. Examples of this phenomenon are documented by [Chu80] and [Jenn77]. This decrease in throughput is due to excessive interprocessor communication during the execution of a single task which causes a saturation effect [Chu78]. We must try to avoid this excessive interprocessor communication in designing an optimal multiple back-end system.

B. The Database Store Interconnection

Another issue is the question of which secondary devices should be connected to which back-ends. For instance, if two back-ends and two disk drives are given, would the configuration shown in Figure 2a where each back-end is connected to exactly one disk drive, be superior to the configuration of Figure 2b where each back-end is connected to both disk drives? The latter configuration is more flexible since any back-end may be employed to execute any user request. However, it is more expensive in terms of hardware complexity. Furthermore, if two back-ends can access the same data, there exists the problem of deciding which back-end should be allowed to access a particular data item.

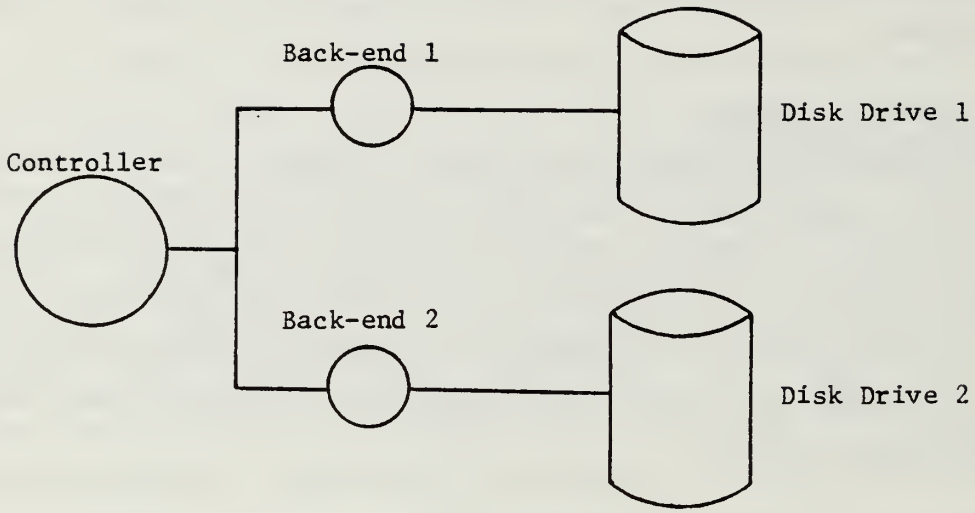


Figure 2a. A Configuration Where Each Back-end is Connected to Only Certain Disk Drives

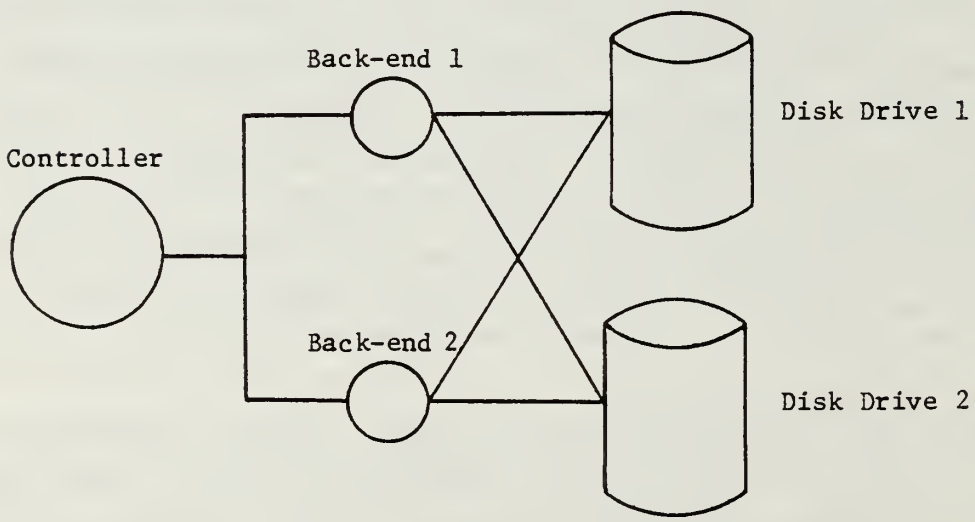


Figure 2b. A Configuration Where Each Back-end is Connected to all Disk Drives

Such a configuration also complicates the concurrency control issue, since one back-end may now read and update the same data being read and possibly updated by another back-end.

1.3.2 System Issues

A. The Data Base Placement

Here, the issue to be resolved is regarding the best way of placing the files constituting the database across the various back-ends in order to achieve the maximum amount of parallel access for the system during read and write operations. In other words, how should the database be partitioned? As much as possible, we should try to ensure that the records constituting the response set to a user request are not stored at a single back-end. Rather, these records in the response set should be distributed evenly across the various back-ends. Such a placement policy should lead to maximal parallel access. Techniques for achieving such a placement policy must be found.

B. The Execution Mode

We ask whether the multiple back-ends should execute in a single-instruction-stream-multiple-data-stream (SIMD) fashion or in a multiple-instruction-stream-multiple-data-stream fashion (MIMD). That is, should each of the back-ends be executing the same request but on different data (SIMD), or should the back-ends be executing in an asynchronous fashion (MIMD). For example, the results of our study in [Meno80] have shown us that MIMD configurations need not always be superior.

C. Directory Structure

Many database management systems also have a secondary body of information (often referred to as the directory) which is used to decrease the search space (and therefore time) of the data base. Issues to be resolved here are whether a directory is indeed necessary and if it is necessary, then what form should it take? Should we use the inverted list organization, the multilist organization, or any of the whold spectrum of alternatives as elucidated in [Hsia70]?

D. The Directory Placement

Having decided on the nature of the directory, we next need to answer the following question. Where is the best place to store the directory? Should it

be stored at the controller, in one of the back-ends, or in all of the back-ends? Should it be duplicated or partitioned? The tradeoffs to be considered are those of storage requirements versus reliability, system throughput and response time.

E. The Security Issue

An important issue that must always be considered in the design of a database system is how security is to be enforced. That is, the question of deciding, in a multi-user environment, who should have what access to which data.

F. The Data Model

Any database management system must decide what data model, e.g., relational, network and hierarchical, it will support. This decision must be made irrespective of whether we have a conventional system, a single back-end system or a multiple back-end system. However, it is possible that some issues peculiar to multiple back-end systems may impact our choice of data model. For example, it will be shown that it is easier to partition the database if it is represented in one kind of data model and not in the other kind. The partitioning criteria is relevant only in the context of multiple back-end database management systems.

G. The Data Manipulation Language

Finally, we must decide upon a language which can be used by the users to manipulate the database in an easy fashion. The choice of a data manipulation language will be closely related to the chosen data model.

1.3.3 Software Issues

A. The Degree of Concurrency

We are designing an architecture that supports multiple users concurrently. At each back-end, we have the choice of processing the requests from these multiple users in an interleaved manner or one at a time. If the requests are not interleaved, then the software in the back-end is simplified. However, the price to be paid in terms of increased user response time and low back-end utilization may be too high. For this reason, we may want to support concurrent request processing at each back-end.

B. Consistency Control and Deadlock Avoidance

If each back-end is to handle multiple requests and multiple transactions, then algorithms must be developed to ensure consistency of the database in the presence of multiple transactions. These algorithms must ensure that each transaction behaves as if it were the only one in the system. Consistency control algorithms may or may not permit deadlocks to occur. If deadlocks are permitted to occur, other algorithms must be developed for detecting and recovering from deadlocks.

1.4 Terminology

We wish to end this section with a brief look at terminology. Throughout the remainder of this report, we will refer to the multiple back-end system which we are attempting to design as the multi-mini database system (MDBS). In MDBS, we will refer to one of the minicomputer systems as the controller which controls the actions of the rest of the minicomputer systems known as back-ends. The throughput of MDBS is defined as the average number of user (program) requests executed by the system in a second. Throughputs may also be defined for various request classes by a straightforward extension of the definition of throughput. The response time of a request in MDBS is the time between the initial issuance of the request by a user (or a user program) and the final receipt of the entire response set of this request by the user (program).

1.5 Contributions of Research

In this dissertation, we present a new approach to the solution of database management problems involving database growth and performance enhancement. A system which uses a multiplicity of conventional minicomputers, novel hardware configuration and innovative software design is presented. This extensible system tries to achieve the ideal goal of having the performance (both response time and throughput) be proportional to the multiplicity of minicomputers.

Our first effort is to identify the problems and bottlenecks involved in developing such an ideal system. Two major problems, one called the controller limitation problem and the other the channel limitation problem are identified. Having identified these problems, our next effort is to systematically eliminate or minimize the ill-effects of these problems. We have also identified a number of other problems.

For studying the multiple back-end database system, we utilize queueing models and simulation. Queueing models and simulation are used at different design stages in

order to aid the design process. Finally, ours is the only comprehensive design of a multiple back-end system that covers all aspects of database management. Algorithms for the four basic request types (insert, retrieve, delete and update), algorithms for aggregate operations, algorithms for performing relational join, algorithms for enforcing security, algorithms for enforcing concurrency control, algorithms for database reorganization and algorithms to be executed at the addition of a new back-end are all analyzed and completely specified.

At a more detailed level, the following contributions may be cited. A solution to the task partitioning problem for multiple back-end database systems has been presented. Unlike previously proposed solutions such as the one in [Dewi78], our solution minimizes message traffic among multiple back-ends thus improving response time and throughput. Experiments have indicated that the proposed solution is extremely effective for a broad range of back-ends. Also, for the first time, the importance of a broadcast capability in multiple back-end database management systems has been clearly demonstrated. More importantly, the very negative impact of not having a broadcast capability has also been demonstrated. Experiments have shown that for systems which do not have a broadcast capability, the response time will improve only with the first few back-ends. After that, the response time actually begins to increase with each additional back-end.

A new and simple algorithm for performing joins on a bus-type architecture is presented and thoroughly analyzed. Unlike other studies, this analysis not only includes I/O time, but it also incorporates the overlap of I/O and join processing. Furthermore, the effects of main memory size on join time are also included in the analysis. Similar analyses of other existing algorithms for join processing are also made and lead to some surprising results. Some other algorithms are actually shown to perform worse with increasing number of back-ends!

The algorithms presented for aggregate operations are not very different from those presented for other systems [Bora80, Schu79]. What is different is the analysis of these algorithms. Once again, a model which incorporates I/O and CPU processing overlap is used. Furthermore, for the first time, an optimum value for the number of back-ends that may participate in an aggregate operation is derived.

Eight schemes are presented for directory management in the new system.

The directory management schemes are different from those for any other system. These schemes are compared in terms of system response time, throughput and storage requirements. A queueing model is used for this purpose. To the author's knowledge, this is the first time that a system has chosen a directory management policy based on analysis using queueing theory.

The queueing network model used is a closed queueing network model. This model incorporates a separate I/O submodel for the disk subsystem. The I/O submodel that we developed was simpler than other existing models [Bard81, Gotl73, Fran74] of disk subsystems in that we were able to obtain a closed form solution for the response time. Our proposed I/O submodel also differs from all other models for disk subsystems in that analysis is made for bulk arrivals. Our analysis assumed bulk arrivals because of the fact that MDBS allows the user to issue requests in a query-based language. As a result, a user query would require a number of records (and, hence, a number of disk tracks) to be accessed at the same time. Finally, a unique iteration technique is employed in order to incorporate this I/O submodel into the overall closed queueing network model.

The data placement policy is based on the work of [Wong71, Roth74]. However, it differs from all these methods in many important respects which will be discussed in Chapter 4. Very briefly, it differs from their work because we are trying to minimize response time in a multiple back-end system and they were interested only in single computer systems. Experiments have been conducted which demonstrate the effectiveness of our placement policy. Certain theoretical results of our optimal placement policy are also presented.

The concurrency control scheme is executed at each back-end rather than at the controller. This is only possible because of the peculiarity of our architecture. In a system like DIRECT [Dewi78], concurrency control algorithms have to be run in the controller. In such a system, the performance of concurrency control algorithms cannot be improved by employing more back-ends. The design of our system has allowed us the flexibility to improve the performance of concurrency control by utilizing more back-ends. We define, for the first time, a term called monolithic consistency to describe the kind of consistency required in a partitioned database (just like inter- and intra-consistency are needed in a centralized database and inter-, intra- and mutual consistency are needed in a distributed database). A solution which preserves monolithic consistency is presented. Our solution is unique in a number of ways. First, it advocates the use of four lock modes, instead of the traditional two lock modes. By se-

parating the insert and delete locks from the update locks, we achieve a greater degree of concurrency. Another contribution is the identification of permutable and compatible requests for a high-level predicate-based query language (and not for a low-level one as in [Gard77]). At a practical level, a method for enforcing locking when using a predicate-based query language is proposed. Unlike [Eswa76] which uses predicate locking, our scheme is much simpler. Unlike [Jord81], the scheme allows predicate-based updates. Unlike both [Eswa76] and [Jord81], the scheme is deadlock free. Hence, it cannot suffer from the 'starvation problem' where transactions are rolled back infinitely and are not guaranteed to complete.

The work on access control and security is based on the work in [MCca75]. However, it differs from [MCca75] in that the method of specifying access privileges is simplified and the security atoms (or clusters) are formed in a different way. We are also able to enforce protection down to the attribute level. A further extension allows for the protection of access against statistical requests. The security-related tables are stored in the multiple back-ends. Each back-end only needs to store a subset of the tables. As a result, the response time and throughput of the entire system is improved. Also, for the first time, we present a comparative study of this security enforcement scheme with others in the literature. Finally, we are able to make the schemes verifiable [Down77]. That is, by guaranteeing the correctness of a small portion of the database management system, we are able to guarantee the correctness of the access control mechanism. The scheme borrows from the ideas of [Down77]. However, [Down77] is a verifiable security mechanism which does not support value-dependent protection. In [Down77], it is speculated that a verifiable mechanism for value-dependent access can not be found. Our scheme contradicts this claim. One final unique feature of our security mechanism is that it is the only one where the refusal of some requested access is guaranteed to lead to a saving in terms of reduced number of accesses to the database store.

The rest of this dissertation will be organized as follows. In Chapter II, we will make a survey of typical software-oriented multiple back-end systems in existence. We shall point out the strengths and weaknesses of these systems and make some recommendations for MDBS so that the weaknesses pointed out for some of the systems can be avoided. In Chapter III, we argue that the attribute-based model is the "superior" or "most appropriate" data model. Accordingly, a simple data manipulation language based on this attribute model is chosen and

presented formally. In Chapter IV, we elaborate on the database placement strategy which was proposed in Chapter II and which was shown to be optimal. Furthermore, algorithms for record retrieval, insertion, deletion and update which make use of this strategy are presented. In Chapter V, we design algorithms for concurrency control which are deadlock free. In Chapter VI, we deal with security-related issues in the MDBS. In Chapter VII, we present a theoretical treatment of the optimal interval at which the database of MDBS should be reorganized in order to take the advantage of the database placement strategy. Also presented is a theoretical treatment of the optimal interval at which additional back-ends may need to be added to the system. At this point, the details of MDBS will have been completely specified. Chapter VIII is then a simulation model of MDBS. Results showing the expected performance of MDBS as a function of the number of back-ends are presented in these sections. In Chapter IX, we will present our final conclusions.

2. A SURVEY OF TYPICAL SYSTEMS AND A STUDY OF SYSTEM ISSUES FOR DESIGN DECISIONS

In this section, we shall survey some of the existing multiple back-end systems. Their relative merits and demerits will be discussed. As we had indicated in Chapter 1, our interest is in software-oriented, multiple back-end systems with a controller. We shall, therefore, include neither distributed database systems such as SDD-1 [Roth80] and Distributed INGRES [Ston76a] nor hardware-oriented systems such as DBC [Bane78b, Kann77b, Kann78]. Based on the findings of the survey, we shall make some design decisions for MDBS. Obviously, in our design for MDBS, we will seek to avoid the weaknesses of the systems surveyed.

The systems that we shall survey in this section are RDBM [Auer80], DIRECT [Dewi78], Stonebraker's machine [Ston78] and DBMAC [Miss80]. Even though many of these systems bear titles which include the word 'machine', they all fall into the category of software-oriented, multiple back-end systems. These four systems do not form a comprehensive list of all systems that fall into this particular category. However, we feel that these four systems are typical of existing software-oriented, multiple back-end systems.

This survey is an important and integral part of the dissertation. We are not presenting, in this survey, all the details of each of the systems surveyed. Rather, for each system surveyed, certain key observations are made. The idea is to point out the problems of these systems so that they will be used as a lesson for better designing of MDBS.

2.1 A Survey of Typical Software-Oriented, Multiple Back-ends

2.1.1 RDBM - A Relational Database System

The RDBM consists of three major components as shown in Figure 3

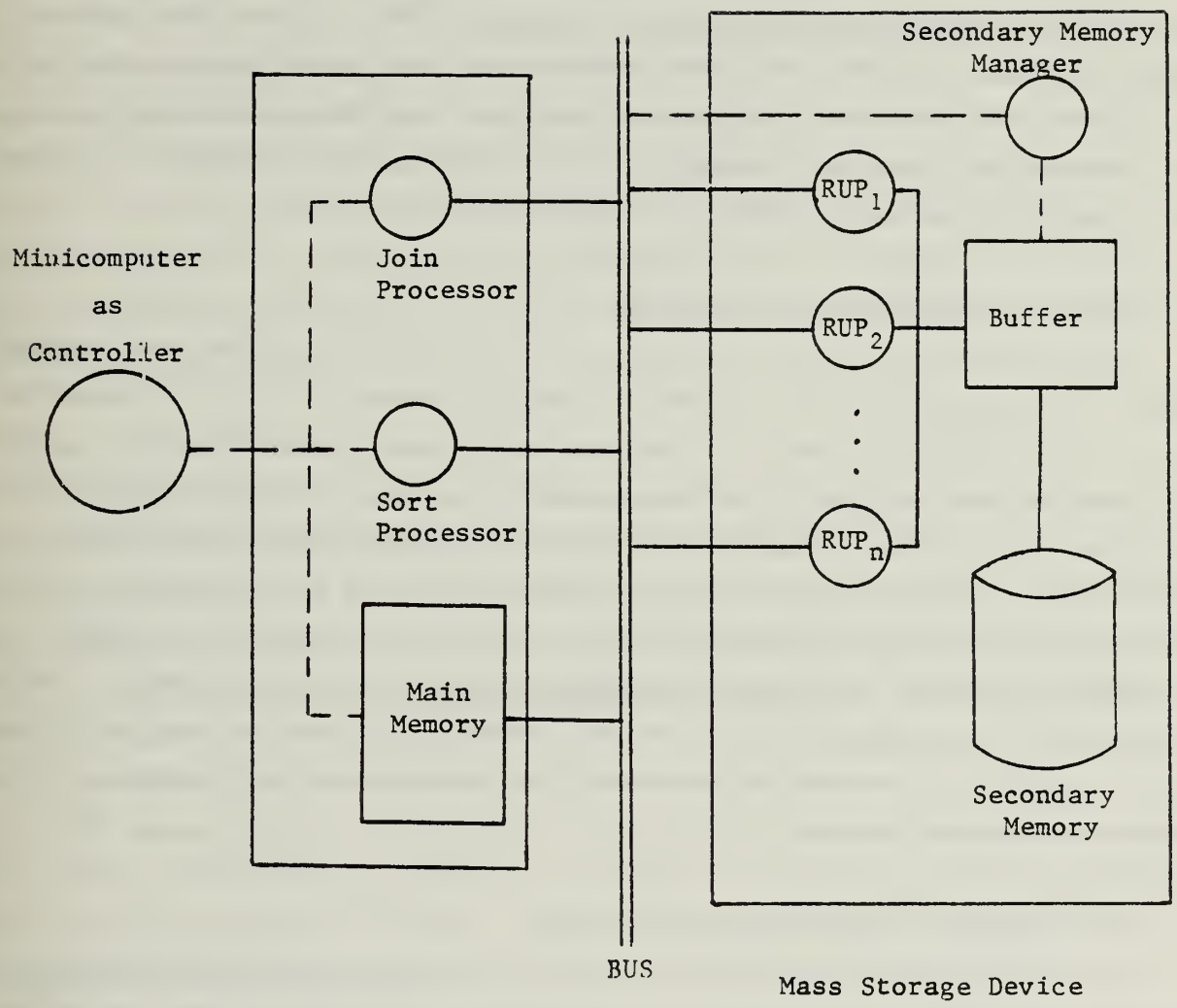
(a) a mass storage device with its own storage manager,

(b) a multiprocessor system consisting of special-function processors working on a large, common main memory, and

(c) a general-purpose minicomputer controlling the different hardware components and performing the preprocessing of the requests.

We shall review each of these three major components in the following.

The mass storage device consists of conventional secondary memories, extended by a block buffer, the secondary memory manager and several processing elements, known as restriction and update processors (RUPs). A retrieve request is exe-



RUP: Restriction and Update Processors

——— Data Lines

– – – – Control Lines

Figure 3. RDBM - A Relational Database System

cuted as follows in RDBM. The pages relevant to the retrieve request are identified and read from the secondary memory to the secondary memory buffer. The records from the buffer are then sent, one by one, to the next available RUP. The RUPs examine the records to determine if they satisfy the criteria of the retrieval request. The RUPs forward the final set of records to the main memory. We note that, at any one time, all the RUPs are executing the same instruction (retrieval request) but on different data (records). A number of observations may be made on this architecture.

A. The Problem of Channel Limitations

First of all, we observe that some of the relevant pages in the secondary memory to be searched on are already in the main memory. This is because the RUPs can only examine records in the secondary memory. Secondly, it is clear that the ultimate limitation in throughput is the rate at which records can be read from the secondary memory to the RUPs via the interconnecting channel. Thus, after a certain point, the use of additional RUPs will not improve the rate at which retrieve (update, delete) requests are executed by the RDBM. We call this problem the channel limitation problem. We would prefer a system which is not limited by the speed of the channel. In other words, the throughput will not be limited by interconnections among the secondary store, the processors and the main memory.

B. The Problem of Software Specialization

The multiprocessor component of RDBM consists of a number of special-function processors - one to perform relational joins, one to perform sorting of retrieved records, etc. In such a system, the workload distribution among the special-function processors could be uneven. For instance, if a large number of user requests requires joins to be performed, but none of these requests requires any sorting to be performed, it is clear that the sort processor will be under-utilized whereas the join processor will be over-utilized. Thus, the best utilization of the multiple processors is not being obtained. Furthermore, such a system may be unreliable. For example, the loss of the join processor will render the RDBM incapable of doing joins. This is known as the software specialization problem.

A system which can continue to perform all the database management functions (perhaps, in a degraded mode) in spite of the loss of a processor is to be pre-

ferred over a system where the loss of a database management function means a permanent denial of that function to the user. Therefore, to overcome this unreliable operation, a system should not have special-function software in the various processors. Rather, it should have general-purpose software in the various processors so that all of them are capable of doing all the database management functions like sorting, joining, etc. In this case, we can also expect a more even distribution of the workload among the processors. In fact, the best possible solution would be to have identical software in all the processors. Then, additional processors may be added to the system with the greatest ease because no new software has to be designed for the additional processors.

C. The Problem of Controller Limitation

The minicomputer in RDBM controls the actions of the various hardware elements in processing a user request. Furthermore, it performs the preprocessing and analysis of the user request to determine the pages in the secondary memory to be retrieved, deleted, etc. This makes the speed of the minicomputer a limiting factor to the throughput of the RDBM. To explain this point, consider, for simplicity, that all user requests require 10 seconds of minicomputer CPU time irrespective of the number of back-ends. This means that RDBM cannot support a throughput rate which is greater than six requests per minute, irrespective of how many processors it uses to speed up operations like join and sorting. We shall, henceforth, refer to this problem as the controller limitation problem.

This problem may also be examined from the viewpoint of the response time. The ideal goal is to achieve a system where the response time improves in proportion to the number of back-ends used in the system. Here, the response time may be considered as the sum of the controller execution time and the back-end execution time. Addition of more back-ends can reduce the back-end execution time, but it cannot reduce the controller execution time which is a constant independent of the number of back-ends. Thus, the controller execution time must be kept to a minimum, if we are to achieve our ideal goal. This leads us to the conclusion that all major tasks must be performed in the back-end processors in a parallel fashion and that the controller must perform minimal work. So, we would like the preprocessing of user requests to be performed by the multiple back-ends in such a way that if there are n back-ends, the total time

for preprocessing is speeded up by a factor of n.

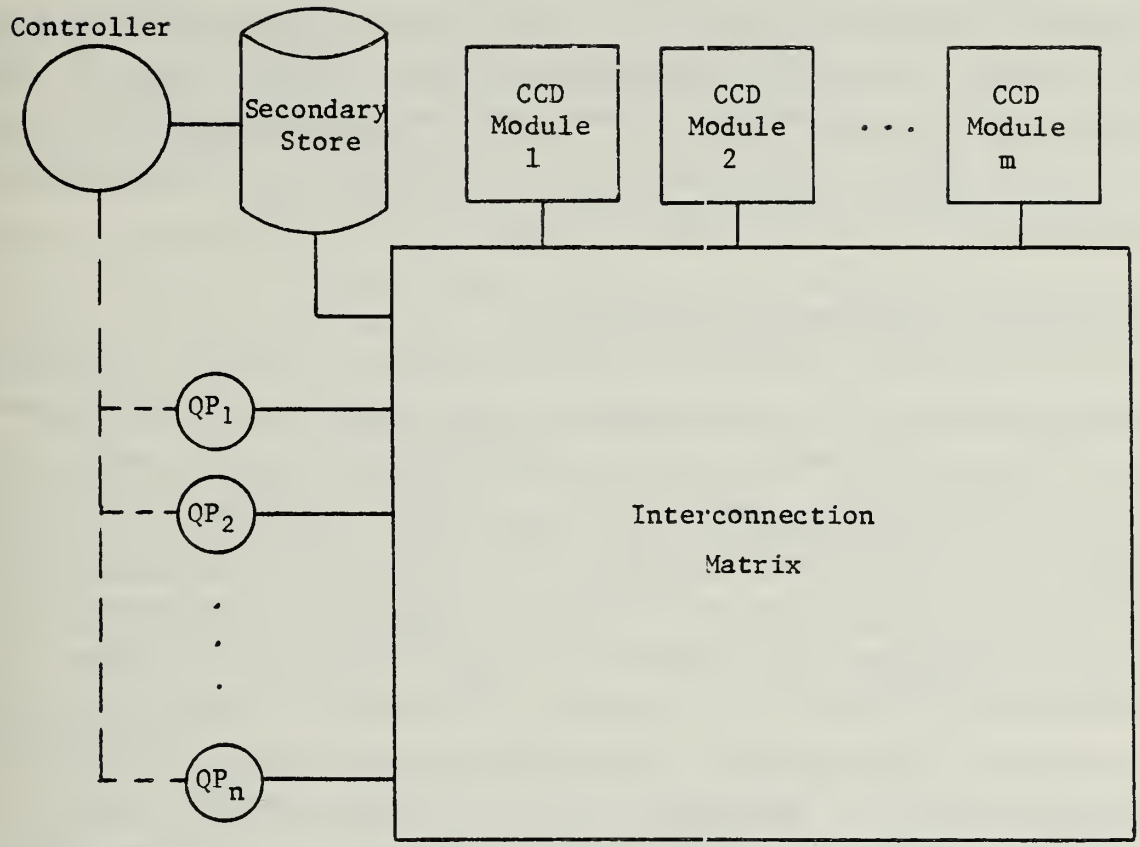
D. The Problem of Data Model Limitation

RDBM supports the relational model of data. Thus, in order to support other data models such as the hierarchical and the network model, it will be necessary to convert the hierarchical and network database to a relational one. Furthermore, requests issued in a hierarchical and network data manipulation language must be translated to requests in the relational data manipulation language. Some researchers have solved the translation problem partially by translating a subset of the network model into the relational model [Katz80]. However, solutions to the problem of translating the entire network model into relational model are not at hand. Thus, the fact that RDBM only supports the relational model constitutes a limitation of RDBM. This is the data model limitation problem. We would prefer to have a data model that is canonical. By a canonical model, we mean that the model allows the entirety of any prevailing data model (i.e., relational, hierarchical and network) to be translated into the data model.

2.1.2 DIRECT - A Multiple Back-end Relational System

As a relational system depicted in Figure 4, DIRECT [Dewi78] consists of four main components: a controller, a set of query processors, a set of CCD memory modules and an interconnection matrix between the set of query processors and the set of memory modules. When the controller receives a user request, it will determine the number of query processors which should be assigned to execute the request. If the relations referenced by the request are not in the CCD memories, the controller will page them in before distributing the request to each of the query processors selected for its execution.

A retrieve request is executed as follows. First, the controller determines the optimal number of query processors that must be utilized to execute the request. This depends upon the size of the relations involved in the request, the priority of the request and the number of currently available processors. The controller then sends the request to each selected processor along with information about the relations to be referenced. The controller also creates a task which waits for a completion signal from each query processor. When all query processors have signalled, the waiting task will transmit the results of the request to the user. For example, let us assume that



——— Data Base

– – – Control Base

QP_i: The i-th Query Processor

Figure 4. The DIRECT System

only a single relation is referenced in the retrieve request. Then, each processor will request the controller for a page of this relation. The controller will search the pages of the relation from the secondary storage, place the pages in the CCD memory and pass the CCD memory address of the requested page to the processor. The query processor may now access the page using the interconnection matrix and perform the necessary work on the page. Finally, the query processor creates a temporary relation containing the selected tuples. This temporary relation will eventually be given to the user.

A. The Problems of Hardware Specialization

We see that DIRECT overcomes the channel limitation problem by use of the interconnection matrix. Strictly speaking, this limitation will be overcome only if each query processor can access any part of the secondary memory. This is not possible in DIRECT, since, only the controller can access the secondary disk memory. The use of the CCD memories as a large cache for the secondary disk memory alleviates the channel limitation problem to a large extent. Ironically, the biggest drawback of DIRECT is its need to use an interconnection matrix whose cost increases as the product of the number of query processors and the number of CCD page frames. While the switching delays of this interconnection matrix do not significantly affect the time to access a page of CCD memory for small number of processors and page frames, such switching delays may, nevertheless, become significant in a full-scale system with many query processors and page frames. Another problem with the interconnection matrix is that it is not easily extensible. For example, the addition of a new CCD page frame will require modifications to the selector interfaces at each query processor, whereas the addition of a new query processor will require modifications to the interfaces at each CCD page frame. Finally, the interconnection matrix is not a conventional hardware element because it must be specially designed. This is the hardware specialization problem.

B. The Problems of Control Message Traffic and Controller Limitation

Each time a new page is needed by a query processor, a message must be sent to the controller and a message must be received from the controller. These two messages may be considered as an overhead for the task of reading a page from the CCD memory. It has been estimated that about 8,000 instructions are needed [Bora81] to send a message from the controller to a query processor or

vice versa. Thus, approximately 16,000 instructions have to be executed before a page may be read from the CCD memory. Assuming that an instruction takes 1 μsec to execute, 16 msecs of overhead are associated with the task of reading a page from the CCD memory. The task of reading a page from the CCD memory only takes 12 msecs [Dewi78]. Thus, the overhead associated with the task of reading a page from secondary memory is of 57%. The above calculation does not include the time taken by the controller to search the relation tables [Dewi78] for the purpose of determining the next page nor does it include the queueing delays suffered by the two overhead messages. Hence, the overhead for the task of reading a page from CCD memory is likely to be greater than the estimated figure of 57%.

Also, the present configuration of DIRECT does not permit broadcasting of requests to the query processors from the controller. As a result, a request which is to be executed by, say, three query processors whould require three separate messages to be sent from the controller to the query processors and this would require approximately (8,000 x 3 =) 24,000 instructions and take up about 24 msecs of controller time. Thus, there is the problem of control message traffic.

Finally, we point out that DIRECT most definitely suffers from the controller limitation problem. That is, the controller is actively involved in many phases of the execution of a request - in the query analysis phase, in the concurrency control operations, in getting the next page, etc. - so that the throughput of DIRECT will be limited by the speed of the controller in the execution of its various tasks.

C. The Problem of Multiple Request Execution

The DIRECT approach does not allow a query processor to support concurrent execution of multiple requests. For instance, consider that while executing a retrieve request, a query processor requests a page which is not in the CCD memory. Then, the query processor is idle until the page is loaded into a page frame in the CCD memory. Such idling could have been avoided, had the query processors been allowed to concurrently execute multiple requests. If each query processor were allowed to concurrently execute multiple requests, more complex software would be required in the query processors. This is the argument used by the designers of DIRECT for not allowing concurrent execution of multiple requests in the query processors. While we understand the ratio-

nale behind this argument, we feel that that was not an alternative. In a multi-user system, the response time is of the utmost importance. By allowing concurrent execution in the query processors, we can improve the response time. Furthermore, a large fraction of the requests is likely to be I/O bound, and the use of concurrent request execution will serve to increase the utilization of the query processors (i.e., the back-ends). Even if much of this anticipated utilization is spent in overhead activities such as task switching, concurrent request execution is likely to provide a performance improvement.

In a system like DIRECT, concurrency control is necessary in spite of the fact that individual query processors do not support concurrent request execution. This is because all the query processors are allowed to access all the pages of the CCD memory. Hence, two query processors may try to update the same page unless concurrency control is enforced. It will be shown that concurrency control is unnecessary, if the back-ends do not support concurrent request execution. Concurrency control in DIRECT is maintained by means of lock tables residing in the controller and requires a number of messages to be exchanged between the controller and the query processors. Furthermore, locking is done at the granularity of a relation and this may reduce the degree of concurrency achievable in DIRECT. As long as no indices are maintained and each request requires the retrieval of entire relations (as in the case of DIRECT), it is difficult to have a finer locking granularity. This is because the two-phase lock protocol [Eswa76] requires that no lock be released until the end of a transaction. Thus, the lock on the first page of a relation cannot be released until the lock on the last page of that relation is set. On the other hand, a non-two-phase lock protocol may have to be used, if a better locking granularity is to be achieved.

Another drawback of DIRECT is that after a query processor completes executing a request, it cannot immediately start executing the next request. This is because no processor maintains a queue of waiting requests. Only the controller maintains such a queue. Consequently, a query processor must first wait until the results of the previous request are received at the controller and then wait till the controller has shipped the next request to the query processor. Two messages of 16 msecs are required between the end of execution of one request and the start of execution of the next. We would prefer to have a queue of waiting requests at each back-end. While such a strategy is not ex-

pected to increase the throughput or decrease the response time dramatically, it will certainly be an improvement over the present strategy of DIRECT. Besides the savings of sixteen msecs we had mentioned, the overhead of maintaining queues has been removed from the controller and passed on to the back-ends. This should contribute to an alleviation of the controller limitation as well. The aforementioned difficulties characterize the problem of multiple request execution.

D. The Problem of Data Model Limitation

Another criticism of DIRECT is the fact that it supports only the relational model of data. As has been previously pointed out, more research needs to be done before hierarchical and network models may be supported by such a system.

In DIRECT, entire relations must be retrieved in order to answer queries. On the other hand, a system which uses index information (like inverted lists on selected attributes) will be able to retrieve relevant portions of relations and save valuable secondary storage access time.

2.1.3 Stonebraker's Machine - A Distributed Database System

A schematic of this system is shown in Figure 5. It consists of a single controller and multiple back-ends. Each back-end is connected to a single disk drive [Ston78]. The controller preprocesses the user queries and performs parsing and decomposition of user queries into requests that access only a single relation. Directory information is stored in one of the back-ends which is designated as the special back-end. After decomposing a query, the controller accesses the directory in this special back-end to determine the back-ends which must be utilized to execute the requests and then sends the requests to these back-ends. After the back-ends return the results of the query to the controller, the controller outputs the results to the user that issued the query.

A. The Problem of the Back-end Limitation

The first thing we note about the distributed database system is that the channel limitation problem does not exist. This is because the multiple back-ends may read data from the secondary storage simultaneously via the multiple channels. However, unlike DIRECT, a request must be executed by one or more specific back-ends. For instance, if a request requires retrieval of information stored in the disk drive attached to the first back-end, only this back-

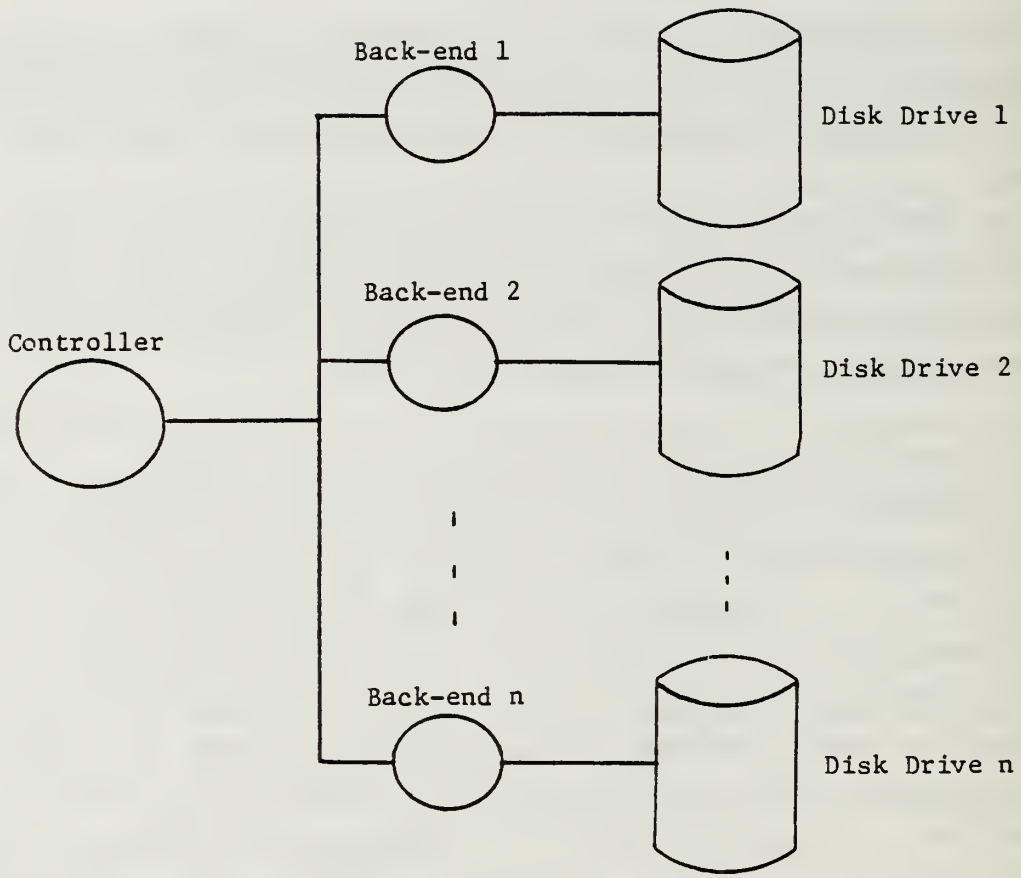


Figure 5. Stonebraker's Machine - A Distributed Database System

end may be employed in order to execute this request. Hence, consider the following situation. Two requests are issued one after another; both require access to a relation stored entirely at the disk drive attached to the first back-end. Then, the second of these requests must wait until the first request completes execution even though many of the other back-ends in the system are idle. This characterizes the back-end limitation problem.

B. The Problem of the Specialized Back-end

The placement of the directory at a specific back-end seems quite unwarrented. Such a scheme requires that, for every request, messages must be sent to and received from the special back-end which carries the directory. The problem of specialized back-end idles the other back-ends of the system.

C. The Problem of Controller Limitation

It should also be pointed out that this system suffers from the controller limitation problem, since parsing and decomposition of queries is done entirely at the controller and will take an amount of time which is independent of the number of back-ends in the system.

D. The Problem of Multiple Request Execution

Another disadvantage of this system is that the back-ends do not support concurrent request execution. As a result, overlap of I/O time with processing of other requests is not possible and back-ends may be idle for large amounts of time waiting for an I/O to complete. Consequently, the best use of processing power is not being made.

E. The Problem of Device Limitation

In this system, each back-end is only connected to a single disk drive. As a result, very large databases of the magnitude, say, of 10^{10} bytes, cannot be supported, since that would require hundreds of back-ends and would make the system extremely expensive. It would seem more reasonable to allow multiple disk drives to be attached to each back-end. This is the device limitation problem.

F. The Problem of Control Message Traffic

The present configuration of the distributed database system does not allow for broadcasting of requests to the back-ends. Hence, a request which requires cooperation among, say, three back-ends would require three separate messages from the controller, i.e., (8,000x3=) 24,000 instrucitons and 24 msecs

of controller's CPU time. The ability to broadcast the requests to the back-ends would save valuable CPU time.

G. The Problem of Data Model Limitation

Finally, a weakness of this system is that it is only designed to support the relational model of data. Furthermore, as in DIRECT, the entire relation has to be retrieved in order to answer an access request for a portion of a relation. As a result, a larger amount of information than is necessary is retrieved from the secondary memory.

2.1.4 DBMAC - An Italian Database System

A view of this system is shown in Figure 6. Since information about this system was obtained through private communication [Miss80], the details are necessarily sketchy. The overall architecture consists of a controller connected to a set of back-end computers over a mass bus. The set of back-end computers is also connected to a set of secondary storage devices via another mass bus. There is no one-to-one correspondence between the back-ends and the secondary storage devices. Each system task (like the request preprocessing task, the concurrency control task, etc.) is performed by a set of modules. The set of modules for a system task are placed in such a way that, as far as possible, no two modules of a task are placed at the same back-end. Thus, all the system tasks are executed by the back-ends in a distributed fashion. Each back-end has a local primary memory and a shared primary memory. Communications among the back-end processors are by means of message passing over the first mass bus.

A. The Problem of Channel Limitation

The first observation of the system that we wish to make is that its throughput is limited by the speed of the second mass bus attached to the secondary memory. This is because even though there are a number of back-ends, they cannot access different portions of the secondary memory simultaneously. In other words, the system suffers from the channel limitation problem. The throughput will also be limited by the speed of the mass bus connected to the controller. In fact, this mass bus will be heavily utilized, since all the system tasks have been broken up into modules that communicate with each other via this mass bus.

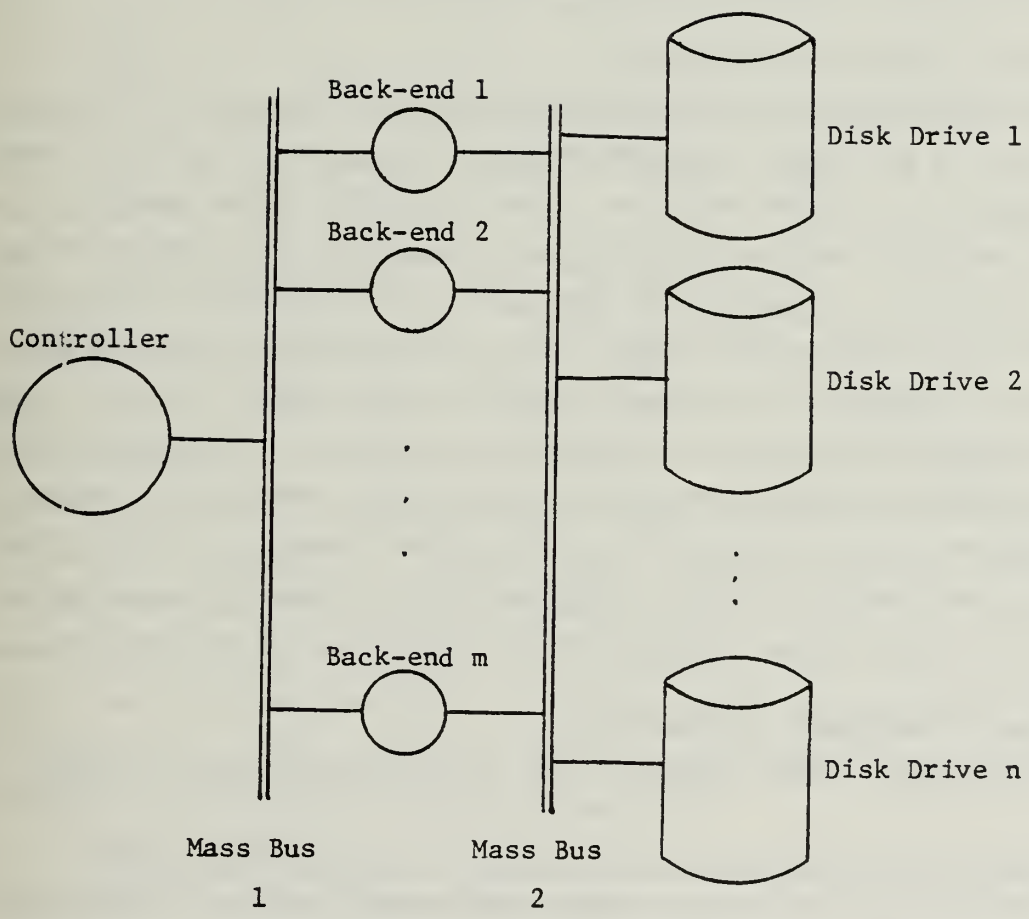


Figure 6. A View of the Italian Database System

B. The Problem of Software Specialization

Since each back-end contains a separate module from each system task, it is clear that the software in the different back-ends is not identical. This leads to a decrease in system reliability, because the loss of one of the processors will render the system incapable of performing any database management function. Furthermore, such a system is not easily extensible. The addition of a new back-end will require the redistribution of the modules among the back-ends which may be a time-consuming and non-trivial task.

C. The Problem of Back-end Limitation

Unlike the aforementioned distributed database system, any of the back-ends of DBMAC may be selected to perform any of the requests, since all back-ends have access to the entire database. Consider the following example. Let us assume that each back-end has enough primary memory to store 100 tracks of information. Also, let us assume that there are n back-ends and n disk drives and that each drive contains 1000 tracks. Then, the probability that a track needed to answer a request is in primary memory is (100/1,000n =) 1/10n (assuming that there is an equal probability of accessing any track). In an organization such as the distributed database system of Stonebraker, however, the probability that a track requested by a back-end is in the primary memory is 1/10. This would cause the back-ends in DBMAC to make many more accesses to secondary memory than the back-ends in the distributed database system of Stonebraker.

D. The Problem of Data Model Limitation

As a final comment, DBMAC supports only the relational model of data.

2.2 Basic Design Considerations for a Multi-Mini Database System (MDBS)

In this section, we will present the overall architecture of a multiple back-end database system, known as MDBS. We will provide the arguments which lead us to this particular architecture. More specifically, we will present the step-by-step development of the architecture. Whenever it is necessary to make a design decision, we shall use simulation studies and analytic techniques to examine the alternatives.

2.2.1 Nine Design Goals

In terms of our survey of typical database systems presented in the previous section, we set nine design goals for MDBS. First, the channel limitation problem should not be present in MDBS in the first place. Second, the controller

limitation problem must be alleviated to as large an extent as possible. Third, the back-ends must execute identical software. That is, all of them must be utilized to perform all the database management functions. The problems of software specialization and back-end specialization can then be eliminated. This leads to increased reliability and to a better workload distribution as has been discussed. It also leads to the simplified addition of more back-ends. Fourth, communications among the back-ends and between the back-end and the controller must be kept to a minimum. Without excessive communications the throughput of MDBS will not taper off after the first few additional back-ends. Consequently, the problem of control message traffic will not exist. Fifth, we resolve not to use any special-purpose hardware in MDBS. As a result, the problem of hardware specialization will not exist. Sixth, we propose to support concurrent request execution in our back-ends in order to eliminate the problem of multiple request execution. For our seventh goal, we resolve to overcome the device limitation problem by attaching more than one disk drive per back-end. Eighth, we will design MDBS in such a way that all the back-ends will participate in the execution of a request. As a result, we will have eliminated the back-end limitation problem. Finally, our ninth goal, we resolve to overcome the problem of data model limitation. In other words, we will propose a canonical data model into which all prevailing data models (such as relational, hierarchical and network) can be fully translated. If these nine design goals are attained, we believe that MDBS may come close to being an ideal system whose reliability, performance (i.e., the throughput and the response time) and growth will be proportional to the number of back-ends employed.

2.2.2 Towards an Ideal System Architecture

In the following section, we will show how we prevent the channel limitation problem from occurring in MDBS. The proposed solution utilizes the technique used in the distributed database system of Stonebraker. By superimposing a data placement strategy on top of that technique, we eliminate the back-end limitation problem in MDBS as well. This strategy is only briefly explained in Section 2.3, since it is expounded in great detail in Chapter IV. Furthermore, the data placement strategy combined with the use of a broadcast capability is shown to prevent the occurrence of the control message traffic problem in MDBS. Next, the device limitation problem is eliminated in MDBS by attaching multiple disk drives to each back-end. In the final section of this Chapter, we will show

that the problem of hardware specialization does not exist in MDBS either. Thus, five of the nine design goals mentioned above are achieved in this Chapter.

In Chapter III, we propose a canonical data model into which all prevailing data models and their data manipulation languages (such as relational, hierarchical and network) can be translated. Thus, we eliminate the data model limitation problem in MDBS. The software to be executed at each back-end is described in Chapter IV. From the discussion in Chapter IV, it will be seen that the problems of software specialization and back-end specialization can be eliminated from MDBS and the goal of using identical software can be achieved. In order to alleviate the controller limitation problem, the directory management, concurrency control and secutity enforcement algorithms are carefully designed. How the careful design of these three algorithms serves to alleviate the controller limitation problem is explained in Chapters IV, V and VI, respectively. The discussion of Chapter V will center on how we choose to eliminate the problem of multiple request execution in MDBS.

The remainder of this chapter is organized as follows. In Section 2.4, we will design a simulation experiment which will illustrate the importance of having a broadcast capability in MDBS. Finally, an overview of the basic design and architecture of MDBS is presented in Section 2.5.

2.3 First Design Decision - Eliminating the Channel, Back-end and Device Limitation Problems

The only architectural decision we have made with regard to MDBS up to this point is that it consists of a controller and a number of back-ends. More decisions regarding the MDBS architecture will be made as we proceed.

Let us now try to design MDBS in such a way as to eliminate any occurrence of the channel limitation problem. It was shown that RDBM and DBMAC both suffered from this problem, whereas, the distributed database system of Stonebraker and DIRECT did not. DIRECT overcame the problem by use of an interconnection matrix which allowed any query processor to access any CCD page frame. The distributed database system of Stonebraker overcame the problem by using a separate disk drive associated with each back-end. Thus, the technique developed for DIRECT and the one developed for the distributed database machine are good candidates for our consideration.

The technique developed for the distributed database system may be attractive owing to its extreme simplicity and low cost. Another strong point

of this technique is that the concurrency control problem is alleviated because two different back-ends will not have the same data item for update due primarily to the use of dedicated disk drives. In spite of these advantages, the technique may be unattractive to us because it suffers from the problem that a given request can only be executed at the back-end attached to the disk drive on which the data relevant to the request resides. This is what we have called the back-end limitation problem. If the data relevant to a request resides on a single disk drive, only a single back-end can be used to execute that request. In other words, the parallelism that is present in the system is not being utilized to execute the request. We would prefer a technique which allows all the back-ends to participate in the execution of a given request

DIRECT is a system which allows all the back-ends to participate in the execution of request. They achieve this by bringing the data relevant to the request into a CCD memory which is accessible from all back-ends. However, such a technique is expensive, since it requires an interconnections matrix whose cost grows as the product of the number of back-ends and the number of CCD memory modules.

We therefore wish to develop a technique which allows all the back-ends to participate in the execution of a request on the one hand and forgoes the costly interconnection matrix on the other hand. Our solution is to have a system with dedicated secondary memories and to store the data in such a way that, whatever the request is, the data to be retrieved for satisfying the request is evenly distributed among the back-ends. Such a data placement strategy allows all the back-ends to participate in the execution of request, by reading data from the secondary memory simultaneously via the multiple channels. Thus, the lack of parallelism due to dedicated devices as exemplified in the distributed database system of Stonebraker does not occur in MDBS. In other words, we have taken the technique of the distributed database system of Stonebraker for overcoming the channel limitation problem and superimposed on it a placement strategy which serves to avoid the back-end limitation problem. An overview of MDBS architecture is depicted in Figure 7. Note that each back-end is attached to multiple disk drives for the elimination of the device limitation problem.

2.3.1 The Need for a Data Placement Strategy

Let us illustrate the strategy first with an example. Consider a file

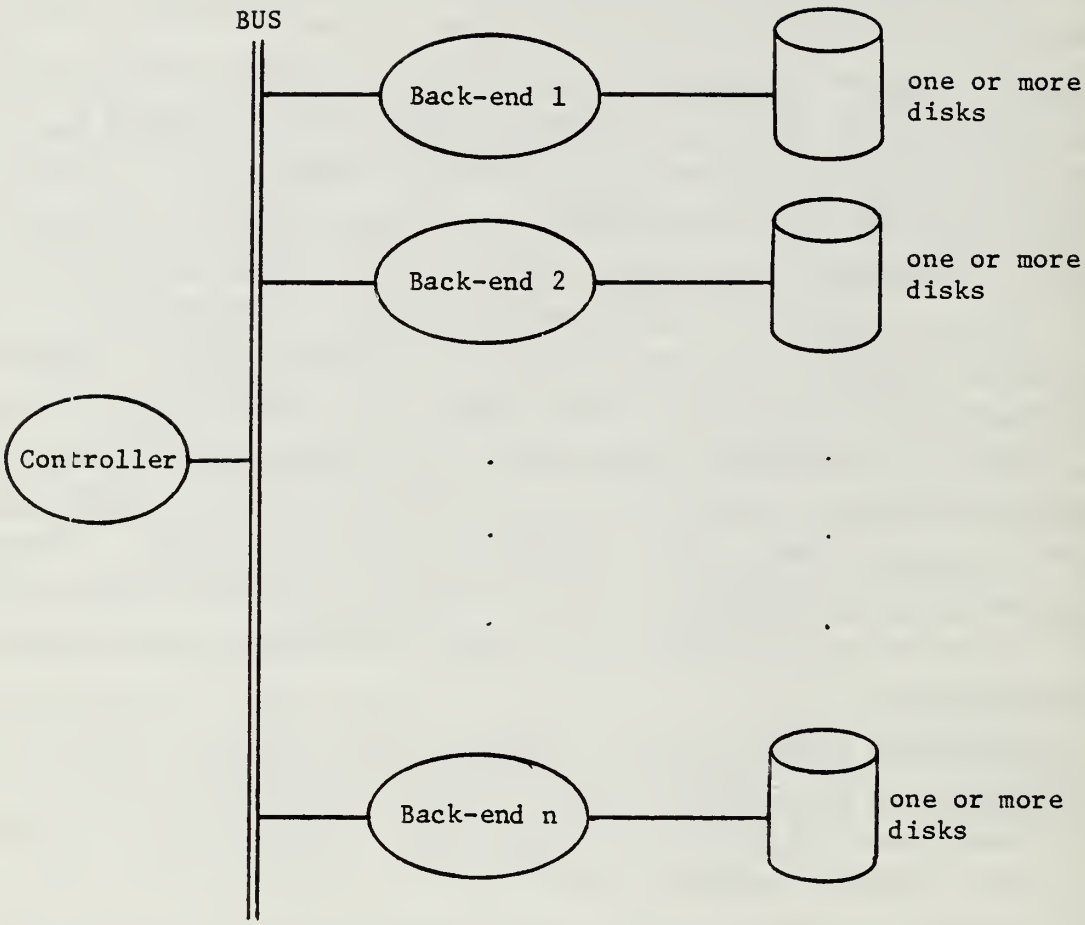


Figure 7. An Overview of MDBS Architecture

with six records as shown in Figure 8. In the figure, an MDBS with one controller and two back-ends is depicted. In order to keep the example deliberately simple, we make the following simplifying assumptions:

(1) Each back-end has only a single disk drive.

(2) Each disk has only three tracks.

(3) All the records are of fixed-length and occupy exactly one track.

(4) Each record contains exactly three keywords.

Consider the arbitrary placement of recrods in Figure 8a, and assume that MDBS receives the following retrieve request.

"Retrieve all records which satisfy the query conjunction (K1&K3)."

The query will be forwarded by the controller to the two back-ends. Given the record placement of Figure 8a, we see that the first back-end must access two tracks, since its disk contains two records - one with keywords K1, K2 and K3, and one with keywords K1, K3 and K4 - which satisfy the conjunction (K1&K3). The second back-end, on the other hand, does not need to access its secondary memory at all, since it contains no records which will satisfy the given query conjunction. Thus, if we let t_d designate the time to access and read-out a track (i.e., seek time and rotation time) and we ignore the directory search time and the time taken by the controller to broadcast the request, then the response time of the request t_q is equal to $2t_d$. Can we do better than this?

In the previous example, the poor response time was caused by an uneven distribution of the data among the two back-ends. One distribution of data which leads to a better response time is the one shown in Figure 8b. The query response time, in this case, is equal to t_d. This is because each back-end has only one record satisfying the given query conjunction and needs to access only one track. Also, the two back-ends can access their respective disks simultaneously. It is this type of parallel operation, combined with the even workload (i.e., data) distribution that contributes to the optimal response time t_q.

We would like to note, however, that the arrangement of records as shown in Figure 8b may cause the response time of some other query (e.g., "retrieve all records which satisfy the conjunction (K1&K5)") to become worse. Thus, the example has demonstrated two things to us. First, a careful data placement strategy must be adopted to obtain improved response time over that which would be obtained with an arbitrary data placement strategy. Secondly, the strategy must be good for all the types of requests which will be issued against the database.

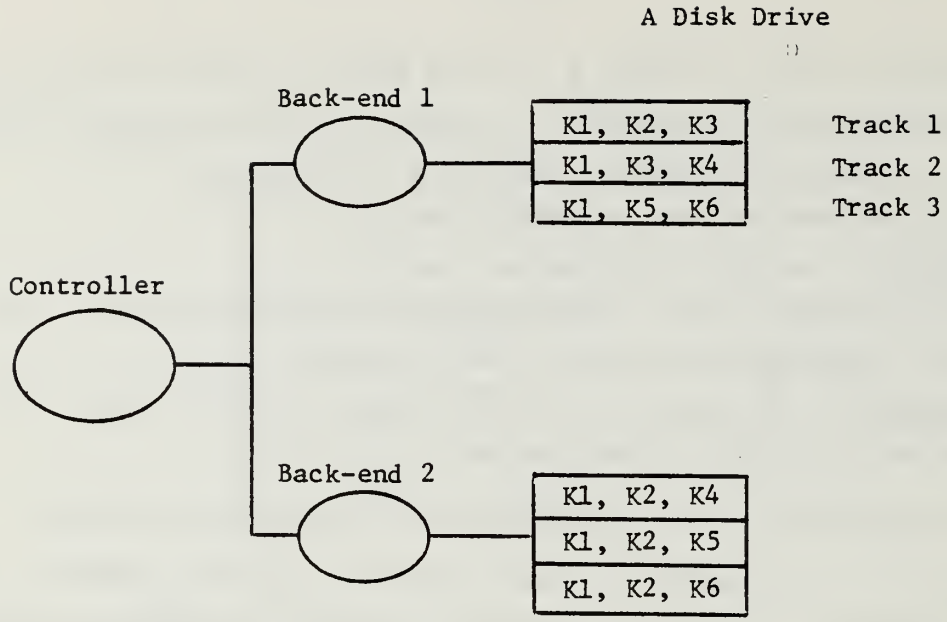


Figure 8a. An Arbitrary Data Placement of 6 Records

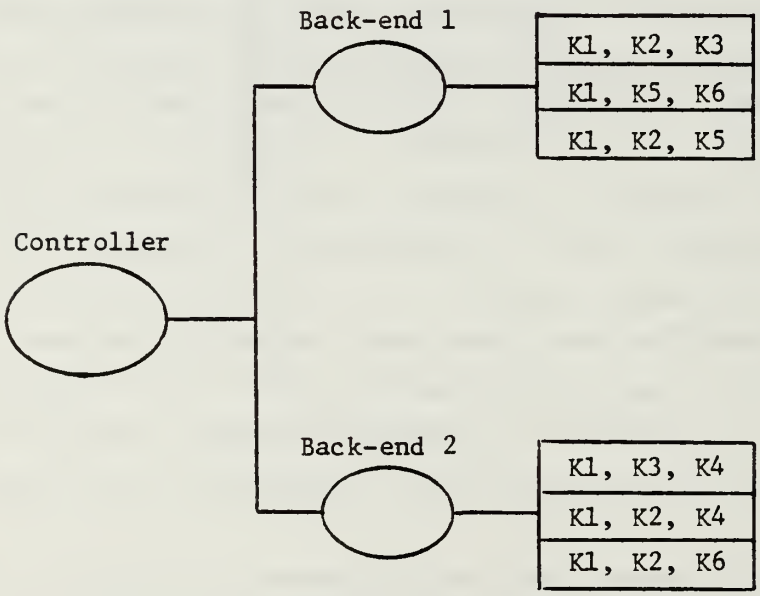


Figure 8b. An Improved Data Placement of Records

2.3.2 An Evaluation of Data Placement Strategies

Three data placement strategies are outlined and evaluated in this section. These are the exact division strategy, the track splitting with placement from the first back-end strategy, and the track splitting with random placement strategy. The differences among these three placement strategies, referred to simply as Strategy A, Strategy B and Strategy C, respectively, are shown by means of an example developed in Figures 9. In Strategy A, the records in the response set were originally divided exactly among the disk drives of the back-ends. Thus, if the response set of a request consists of five tracks of data and there are three back-ends as depicted in Figure 9a, each back-end will contain (5/3=) 1.67 tracks of data. Strategy B also consists of dividing the records of a response set equally among the back-ends. However, it is different from Strategy A in that the division of the response set takes place at tract boundaries. Thus, if the response set of a request consists of five tracks of records and there are three back-ends, the disk drive of each back-end will contain one track of data. The remaining two tracks are then assigned to disk drives of the first two back-ends. In other words, the disk drive of the first back-end contains two tracks of the response set, the disk drive of the second back-end contains two tracks of the response set and the disk drive of the third back-end contains only one track of the response set as illustrated in Figure 9b. Strategy C is similar to Strategy B in that the division of the response set takes place at track boundaries. It differs from Strategy B in that the left-over data after exact division of the data in terms of tracks among the back-ends are assigned to the disk drive of arbitrary back-ends. Thus, in the example where the response set of a request consists of five tracks of records and there are three back-ends, the disk drive of each back-end will contain initially one track of data. The remaining two tracks are then assigned to the disk drives of the two back-ends picked randomly and not necessarily to the disk drives of the first two back-ends as in Strategy B. The situation is depicted in Figure 9c.

Four simulation models of MDBS are developed using SIMULA on a DEC System 20. First, we test out the simulation model in which there is no data placement policy. That is, no assumption is made regarding where the response set of a request is located. One or more back-ends may be employed in the execution of a request depending upon where the response set to the request is stored. Parallelism will be utilized only if more than one back-end needs to be employed. Such a simulation model would approximate a system like the distributed database system of Stonebraker [Ston78], where no special placement

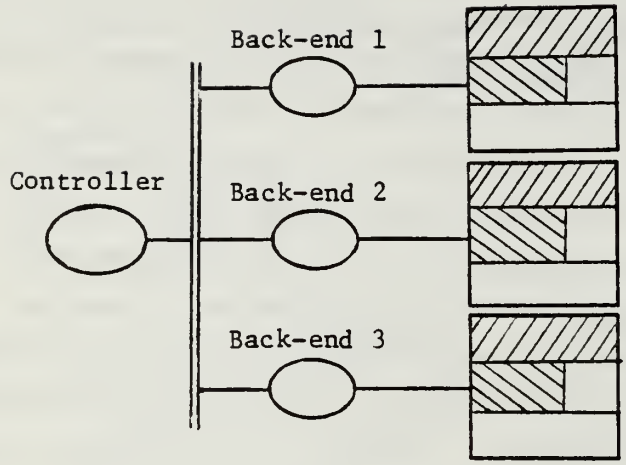


Figure 9a. Strategy A - Exact Division of Data by the Number of Back-ends for Placement

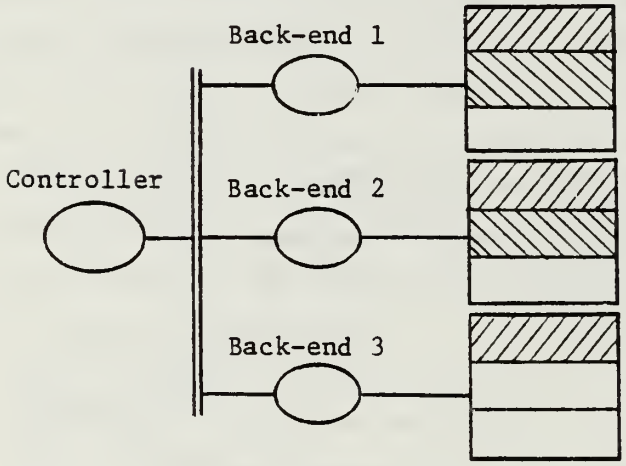


Figure 9b. Strategy B - Back-ends Accommodating Only Track-size Data with the First Back-ends Accommodating the Extra Ones

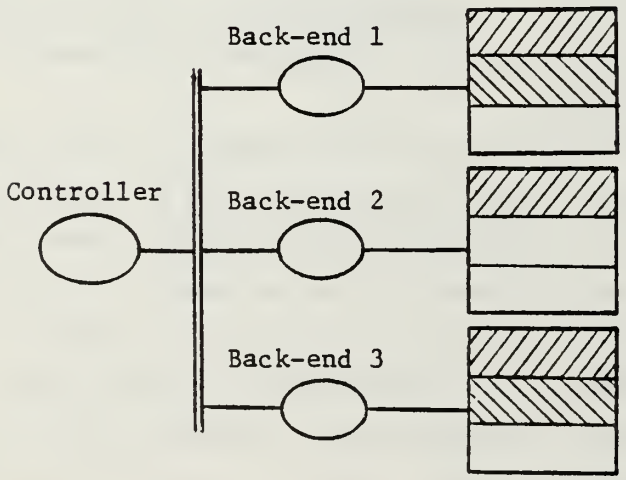


Figure 9c. Strategy C - Back-ends Accommodating Only Track-size Data with Arbitrary Back-ends to Picking up the Extra Ones

Figure 9. Three Different Data Placement Strategies

policy is employed. In the simulation model, a random number generator is used to determine how many back-ends will participate in answering the request and the request is then sent to these many back-ends. The remaining three simulation models simulate MDBS under the three aforementioned placement strategies.

For all four simulation models, each back-end has a queue of requests which are executed in a first-in-and-first-out basis. Also, the time to broadcast a request and the time to return the results to the controller are ignored, since we are not interested in modelling message traffic at this point (that will be done later). Furthermore, at a back-end, the time taken to retrieve records from the secondary memory is assumed to be dominating the execution time of a request. Thus, CPU execution time is ignored. Finally, in all four models we make the assumption that a request is never satisfied by the data that is already in the memory buffers of the back-end. The request always requires the data to be retrieved from the secondary memory. While such an assumption implies a worse case situation, it has the advantage that factors such as good buffering techniques will not affect our results.

The results of our simulation studies are tabulated in Tables I and II. It is assumed that 25% of the requests generated are 'insert record' requests and the remaining are delete, update and retrieve requests. Each of the three latter types of requests require retrieval of information from disks. It is also assumed that anywhere between five and twenty tracks of information will have to be retrieved and searched by all the back-ends in order to answer a retrieve, delete or update request. If more than one track has to be retrieved by a back-end, no assumption is made that these tracks in the secondary memory are sequentially next to each other. For example, consider an MDBS system with three back-ends and assume that the response set of a request consists of six tracks of data. Then, the data placement strategy will ensure that each back-end stores two of the six tracks of the response set. However, the two tracks of the response set that are placed at a back-end are assumed to be randomly stored on its disks. In Section 2.3.4, we will present another simulation study in which we will assume that if more than one track has to be retrieved by a back-end, then these tracks are sequentially placed on the disk tracks of that back-end. Finally, the requests are assumed to arrive in a Poisson stream. This assumption implies that each request is independent of all others. We have also simulated the systems assuming different arrival patterns, and the

Strategy / Inter-Arrival Time of Requests (msecs)	Number of Back-ends = 15			
	No Placement Strategy	Strategy A	Strategy B	Strategy C
100	255	75.1	64.8	38.2
200	208	57.4	52.6	33.1

Strategy / Inter Arrival Time of Requests (msecs)	Number of Back-ends = 10			
	No Placement Strategy	Strategy A	Strategy B	Strategy C
100	371	119	94.9	65.0
200	269	78.7	69.3	51.6

Table I. The Response Time (in msecs) of MDBS Under Various Data Placement Strategies

	The ratio = $\frac{\text{Response time with no placement strategy}}{\text{Response time with best placement strategy}}$	
I \ N	10	15
100	5.71	6.67
200	5.21	6.28

N: Number of Back-end

I: Inter Arrival Time in Milliseconds

Table II. The Improvement Caused by a Good Placement Strategy

results do not differ significantly from the ones with Poisson arrivals. Similarly, simulations have been run with different percentages of insert requests. Again, they do not seem to affect the results significantly and their results are therefore not presented herein. As a final note, the method of subruns [Fran77] was used to make sure that the results were unbiased - that is, to take care of correlated observations in the simulation.

The actual response times of MDBS under the various placement strategies are shown in Table I for various request interarrival times and for various number of back-ends. The first observation we make from the results is that the performance of MDBS can be improved by the use of a placement strategy. Among the various placement strategies, Strategy C is the best and Strategy A is the worst and this may be explained intuitively as follows. Consider a system with three back-ends and assume that a particular request's response set consists of 195 records. Furthermore, let us assume that 32 records will fit into a single track. In all the three strategies, the disk drive of each back-end would have stored two tracks of records from this response set, making a total of 192 stored records. The remaining three records would have been stored in the disk drives of the three back-ends on the basis of the placement strategy used. Strategy A ensures that the disk drives of each of the back-ends will contain exactly one of these three left-over records. Strategies B and C, on the other hand, ensure that all the three left over records are placed in a single track of the disk drive of one of the back-ends. Strategy A is no better than Strategies B and C because the time to retrieve one record from the secondary memory is almost the same as the time to retrieve three records from the secondary memory. This is because the minimal disk access time is the time to access a track. Let us denote the time to retrieve an entire track of records from the secondary memory as x. Then, in Strategy A, each back-end will spend 3x time units for this request. On the other hand, in Strategies B and C, only one of the back-ends will spend 3x time units for this request. The other two back-ends will spend only 2x time units for this request. This is the reason for the improved performance of Strategies B and C.

The reason why Strategy B performs worse than Strategy C may be explained as follows. In Strategy B, after dividing the tracks of a response set equally among the back-ends, the extra, say, i tracks are assigned to the first i back-ends. As a result, back-end #1 will always take the longest time to answer a retrieve (delete or update) request. The response time for this request is

proportional to the time taken by back-end #1 to answer the request, since it always takes the longest time. In general, back-end #1 will do more work and take more time than any of the other back-ends, back-end #2 will do more work and take more time than back-ends #3, #4, #5, and so on. In Strategy C, however, the workload distribution is more even owing to the fact that after initial distribution the excess tracks are assigned randomly to the back-ends. Hence, no one back-end does more work and take more time than any other.

In order to emphasize the advantages of a good placement strategy, i.e., Strategy C, over one where no strategy is used, in Table II, we tabulate the response time ratios in these two cases (the ratios are calculated from data in Table I). It is seen that the response time can be improved by a factor of as much as 6.67 with good data placement. Furthermore, it is seen that for larger number of back-ends, the ratio is larger. This is an interesting result, since it tells us that the effect of our proposed placement strategy will become more evident at larger of number of back-ends. As we are trying to develop an extensible system, such a result is encouraging. It implies two prospects: (1) The response time of MDBS will be better than the response time of a system that does not use a 'good' data placement strategy. (2) The response time of MDBS will improve as the number of back-ends of MDBS is increased. The greater the number of back-ends, the better the response time.

2.3.3 An Evaluation of the Data Placement Strategies Using More Refined Assumptions

In the previous simulation study, we assumed that the data constituting the response set were evenly distributed among all the back-ends. We also assumed that the data of the response set at any one back-end were randomly placed and not necessarily next to each other on the disk. We now focus on more refined simulation by making the assumptions more realistic. Since a large amount of data being read and manipulated by a back-end is likely stored on the disk sequentially, we will not assume that they are placed randomly on the disk. Instead, we assume now that they are likely placed sequentially, i.e., one track followed by another track. We also use more refined blocking factors. Instead of dividing data into tracks, we now assume that the data will be divided into smaller units, known as pages. Let there be n back-ends in MDBS. Also, let a request require, on the average, the retrieval of x records. Finally, let these x records be stored as s groups of x/s records each. The value of s determines the amount of sequentiality that is present in the data being re-

trieved. For example, in the extreme case, s is of value one. This implies that all the data being retrieved is stored sequentially, since they are in one group. The larger the value of s, the greater the randomness with which the records being retrieved are scattered over the disk tracks at a back-end.

For data placement, the following three new strategies are considered.

(1) Place $\frac{x}{sn}$ records from each group at every back-end. That is, $g(\frac{x}{sn})$ records of a group are placed in some of the back-ends and $h(\frac{x}{sn})$ records are placed in the remaining back-ends, where g(y) stands for the nearest integer equal to or greater than y, and h(y) stands for the nearest integer equal to or less than y.

(2) Let a page accommodate p records. Then, the $\frac{x}{s}$ records of a group are stored in $g(\frac{x}{sp})$ pages and each back-end receives $\left(\frac{g(\frac{x}{sp})}{n}\right)$ pages. That is, certain back-end receives $g\left(\frac{g(\frac{x}{sp})}{n}\right)$ pages and certain other back-end receives $h\left(\frac{g(\frac{x}{sp})}{n}\right)$ pages.

(3) Let a track store t records. Then, the $\frac{x}{s}$ records of a group are stored in $g(\frac{x}{st})$ tracks. Each back-end receives $\frac{g(\frac{x}{st})}{n}$ tracks of data. That is, certain back-end receives $g\left(\frac{g(\frac{x}{st})}{n}\right)$ tracks of data and certain other receives $h\left(\frac{g(\frac{x}{st})}{n}\right)$ tracks of data.

After initial placement of data, the remaining data will be placed in the following way. For Strategy 2, some i of n back-ends receive $g\left(\frac{g(\frac{x}{sp})}{n}\right)$ pages and the remaining (n-i) back-ends receive one page less. The i back-ends which receive one page more may be either the first i back-ends or any i consecutive back-ends starting from a randomly chosen back-end which gives rise to two variations of Strategy 2. We do not consider other possible variations of Strategy 2 and leave them for future research. Similarly, two corresponding variations of Strategy 3 are also considered.

Strategy 1 is the same as Strategy A discussed in Section 2.3.2. Also, the two variations of Strategy 3 are similar to Strategies B and C of Section 2.3.2. The two variations of Strategy 2 are the counterparts of Strategies B and C of Section 2.3.2 where the division is done at page boundaries rather than at track

boundaries.

Five separate simulation models of MDBS, one for each of the aforementioned data placement strategies, are designed and evaluated using SIMULA on a DEC System 20. The assumptions made in these simulations are similar to the ones made in the simulation of Section 2.3.2 and will not be repeated here.

2.3.3.1 The Choice of a Superior Strategy for Data Placement on the Basis of Better Response Time

In discussing the new simulation results, we shall refer to Strategy 1 as the exact division strategy; Strategy 2 as the page splitting strategy. The two variations of Strategy 2 are called page splitting with placement from the first back-end and page splitting with random placement, respectively. Similarly, the two variations of Strategy 3 are referred to as track splitting with placement from the first back-end and track splitting with random placement. The number of records needed to be retrieved for a request depends upon whether the size of the request is small, or large. A small-sized request is one which requires the retrieval of one to 320 records. A medium-sized request is one which requires the retrieval of between 320 and 1,280 records. Large-sized requests require the retrieval of between 1,280 and 6,400 records. We note that 64 records can fit in a track. Thus, the retrieval of between 320 and 1,280 records is equivalent to the retrieval of between 5 and 20 tracks of records.

The number of groups into which the retrieved records fall is chosen from the set {1,5,20}. Choosing one as the number of groups implies that all the data are sequentially related. Thus, they form a single group. Choosing 20 as the number of groups implies on the other hand that there are 20 random data aggregates, although data in the individual aggregates may be sequentially related. Another parameter which is varied is the number of back-ends involved. This is chosen from the set {5,10,15}.

Tables IIIa, IIIb and IIIc, show the simulation results for small-sized, medium-sized and large-sized requests, respectively.

The results indicate that the track-splitting-with-random-placement strategy is the best one over all possible numbers of back-ends, of groups and of of the request size. Only when the number of groups is one does the superiority of this strategy become unclear. Setting the number of groups to one implies

Number of records retrieved is between 1 and 320

Inter-arrival time = 200 msecs

Small-Sized requests

Response time results in msecs

NUMBER OF GROUPS = 1

Number of Back-ends	Exact Division	Page Splitting Round Robbin	Page Splitting Random	Track Splitting Round Robbin	Track Splitting Random
5	60.4	40.1	40.9	39.4	40.3
10	36.6	36.3	36.9	38.9	38.7
15	35.3	35.1	36.1	38.8	37.8

Inter-arrival time = 200 msecs

NUMBER OF GROUPS = 5

Number of Back-ends	Exact Division	Page Splitting Round Robbin	Page Splitting Random	Track Splitting Round Robbin	Track Splitting Random
5	358	343	343	501	89.8
10	338	317	298	485	66.8
15	332	306	252	485	57.6

Inter-arrival time = 1000 msecs

NUMBER OF GROUPS = 20

Number of Back-ends	Exact Division	Page Splitting Round Robbin	Page Splitting Random	Track Splitting Round Robbin	Track Splitting Random
5	977	929	726	1250	266
10	969	886	421	1230	159
11	961	877	305	1230	130

Table IIIa. Response Time Results for the Various Strategies for Small-Sized requests

Number of records retrieved is between 320 and 1280

Inter-arrival time = 200 msecs

Medium-Sized requests

Response time results in msecs

NUMBER OF GROUPS = 1

Number of Back-ends	Exact Division	Page Splitting Round Robbin	Page Splitting Random	Track Splitting Round Robbin	Track Splitting Random
5	75.9	75.6	77	74.8	75
10	52.8	52.6	53.4	50.6	51.2
15	45.8	45.5	47.4	44.6	46.9

Inter-arrival time = 200 msecs

NUMBER OF GROUPS = 5

Number of Back-ends	Exact Division	Page Splitting Round Robbin	Page Splitting Random	Track Splitting Round Robbin	Track Splitting Random
5	617	597	680	535	244
10	432	421	420	504	143
15	389	379	379	494	109

Inter-arrival time = 1000 msecs

NUMBER OF GROUPS = 20

Number of Back-ends	Exact Division	Page Splitting Round Robbin	Page Splitting Random	Track Splitting Round Robbin	Track Splitting Random
5	1060	1050	1000	1250	246
10	1010	996	987	1230	159
15	996	969	868	1230	131

Table IIIb. Response Time Results for the Various Strategies for Medium-Sized Requests

Number of records retrieved is between 1280 and 6400

Inter-arrival time = 500 msecs

Large-Sized requests

Response time results in msecs

NUMBER OF GROUPS = 1

Number of Back-ends	Exact Division	Page Splitting Round Robbin	Page Splitting Random	Track Splitting Round Robbin	Track Splitting Random
5	238	237	245	235	242
10	120	119	121	119	121.0
15	87.1	86.9	91.1	86.2	92.2

Inter-arrival time = 500 msecs

NUMBER OF GROUPS = 5

Number of Back-ends	Exact Division	Page Splitting Round Robbin	Page Splitting Random	Track Splitting Round Robbin	Track Splitting Random
5	517	513	571	491	465
10	308	305	312	290	265
15	258	256	266	247	218

Inter-arrival time = 1000 msecs

NUMBER OF GROUPS = 20

Number of Back-ends	Exact Division	Page Splitting Round Robin	Page Splitting Random	Track Splitting Round Robin	Track Splitting Random
5	1580	1560	1540	1360	866
10	1220	1210	1240	1270	435
15	1130	1110	1090	1250	307

Table IIIc. Response Time Results for the Various Strategies for Large-Sized Requests

that all the data in the database must be stored sequentially. This is a rare occurrence in database systems. Should this happen, all the strategies are approximately equal in terms of the resulting response times. However, the track-splitting-with-random-placement strategy leads to very dramatic improvements in response time over the next best strategy for the larger number of groups. The superiority of this strategy over all other strategies is most evident for small to medium-sized requests and when the data tends to be more randomly distributed. Situations where the average response time of MDBS using the next best strategy is five times slower than the response time of MDBS using the track-splitting-with-random-placement strategy are noticed. For instance, for small-sized requests, when the number of groups is five and the number of back-ends is fifteen, the track-splitting-with-random-placement strategy leads to an average response time of 57.6 msecs. The nearest rival, which is the page-splitting-with-random-placement strategy, leads to a response time of 252 msecs which is almost five times slower. The track-splitting strategy performs better than the page-splitting strategy for exactly the same reason as that Strategy C of Section 2.3.2 performs better than Strategy A of the same section.

Thus, from a performance point of view, the best data placement strategy is the one where the response set is divided up into tracks of data and stored as multiples of data tracks. Should there be extra data tracks, after even distribution among the back-ends, they are assigned to the disks of consecutive back-ends starting from a randomly selected back-end.

In Chapter IV, we will relate the notion of clusters with the notion of groups. Clusters are formed on the basis of their attributes and their potential utilization. However, the placement of the clusters is related to the placement strategies of the groups which relies on the present simulation studies.

2.3.3.2 The Choice of a Superior Data Placement Strategy on the Basis of Better Storage Utilization

Let us now consider the various strategies from the point of view of storage utilization. In this case, the comparison is between a strategy which stores groups of records in multiples of tracks (as in Strategy 3) and a strategy which stores groups of records in multiples of pages (as in either Strategies 1 or 2). Let us call the former the track-splitting strategies and the latter the page-splitting strategies.

There are two factors affecting storage utilization: First, there is the

un-utilized space owing to the fact that records do not exactly fit in a page (or a track). Thus, if a page can hold only 512 bytes and each record is of size 200 bytes, 112 bytes on each page is wasted. The second factor that leads to un-utilized space is the fact that each group of records ends on page boundaries (or track boundaries) and records from two different groups are never placed on the same page (or track). The first factor is favorable to track-splitting strategies and the second to page-splitting strategies.

Consider again that there are x records to be retrieved and that they are to be retrieved as s groups of $\frac{x}{s}$ records each. In the page-splitting stragety, the $\frac{x}{s}$ records are stored in $g(\frac{x}{sp})$ pages. The wasted space in $g(\frac{x}{sp}) - 1$ of these pages is the difference between the page size and the p record sizes. The wasted space on the last page is

$$\text{page size} - (\frac{x}{s} - (g(\frac{x}{sp}) - 1)p) \times \text{record size}$$

Let page size = p', record size = r, and track size = t'. Then, percentage of wasted space in the page splitting strategy is

$$\frac{(g(\frac{x}{sp})-1)(p'-pr) + p' - (\frac{x}{s} - (g(\frac{x}{sp}) - 1)p)r}{g(\frac{x}{sp})p'}$$

Similarly, the percentage of wasted space in the track-splitting strategy is

$$\frac{(g(\frac{x}{st}) - 1)(t' - tr) + t' - (\frac{x}{s} - (g(\frac{x}{st})-1)t)r}{g(\frac{x}{st})t'}$$

The percentage of wasted space will depend on the size of a record and the number of records of a group. Some experimental results are shown in Table IV. The record size is chosen from the {200, 300, 400} bytes, and the value of $\frac{x}{s}$ is chosen from the set {20, 50, 100}. The results are as follows. In either strategy, the wasted space decreases when the record size is increased. The page splitting-strategy has less wasted space when the amount of sequentiality in the data being retrieved is low. However, if a good clustering policy is employed so that more of the records that are likely to be retrieved together are stored together, then the track-splitting strategy will waste less space. In Chapter IV, we will present such a policy. In conclusion, we shall use the track-splitting strategy in MDBS both for the good response time and for the low storage wasted.

PAGE-SPLITTING POLICY

Record Size (bytes)	Number of Records Per Group		
	30	60	90
200	.219	.219	.219
300	.414	.414	.414
400	.219	.219	.219
500	.023	.023	.023

TRACK-SPLITTING POLICY

Record Size (bytes)	Number of Records Per Group		
	30	60	90
200	.625	.023	.023
300	.438	.023	.023
400	.250	.023	.023
500	.063	.023	.023

Table IV. Comparison of Storage Wastage Between Two Different Placement Policies are Used.

2.3.4 Next Step in the Design Process

We have now gone one step further in our design of MDBS. We have decided that it is going to be a dedicated system in which each back-end has attached to it a number of disk drives which are not accessible from any other back-end. As a result, the controller and device limitation problems are eliminated in MDBS. Furthermore, we have decided to use a data placement strategy in order to store the data in the disk drives in an evenly distributed manner for the improvement of the response time and storage utilization. Such a strategy will be proposed in details in Chapter IV. We also argue that such hardware configuration and data placement will ensure that all back-ends will participate in the execution of a request and eliminate the back-end limitation problem. Unlike DIRECT, we do not need a costly interconnection matrix in order to achieve this. Finally, we will show in the next section that as a result of our data placement strategy the amount of control message traffic needed in MDBS is much less than in DIRECT. We will also propose, in the next section, the incorporation of a broadcast capability in MDBS for further minimizing control message traffic.

2.4 Second Design Decision - Minimizing the Problem of Control Message Traffic

Consider a DIRECT system with three query processors executing a request that requires access to nine pages of memory. The execution of such a request requires three messages to be sent from and to be received by each of the three query processors. It also requires the controller to send and receive nine messages. The figures are only for messages that are sent and received by the processors asking for page frames and do not include the three messages that must be sent by the controller in order to initiate the query processors nor does it include one message from each of the query processors when they output the results. In all, DIRECT exchanges 24 messages while executing this request for nine pages.

MDBS, on the other hand, will require only six messages. Three messages are needed to send the request to the three back-ends and one message will be received from each of the back-ends when they output the results.

The number of messages needed in MDBS may be further reduced to four if we have a broadcast capability. Our feeling that a broadcast capability is important for MDBS is prompted by the following. Since the data placement strategy ensures that all back-ends will be participating in answering a request, the

request must now be sent to every back-end. Thus, with a broadcast capability of a system of n back-ends, we need only a single message rather than n messages to broadcast a request. The simulation experiment in the next section illustrates, graphically, the advantages of a broadcast capability.

2.4.1 The Need for a Broadcast Capability

Two sets of simulations are run - one for MDBS without the broadcast capability and one for MDBS with such a capability. Once again, the simulation programs are written in SIMULA on a DEC System 20. It is assumed, as in the earlier simulation experiments, that four types of requests (i.e., retrieve, insert, delete and update) are issued and that 25% of all requests are of the insert type. Also, as before, it is assumed that anywhere between five and twenty tracks will have to be retrieved and searched in order to answer a retrieve, delete or update request. The requests are assumed to arrive in a Poisson stream. Finally, it is assumed that the time taken to retrieve records from the secondary memory dominates the CPU processing time so that the latter time may be ignored. Simulations are run for various number of back-ends and various request inter-arrival times. The response time results are tabulated in Table Va.

Table Va indicates that the inclusion of a broadcast capability can make the MDBS system up to 2.5 times as fast as it would be without such a capability. Also, the utility of such a capability becomes more evident for larger number of back-ends. This is, of course, to be expected. It should be pointed out that the improved response time in the case of MDBS with the broadcast capability is a direct result of this capability, since the control message traffic is reduced.

2.4.2 An Evaluation of the Broadcast Capability with More Refined Assumptions

The results of Table Va assumed that the tracks to be retrieved from the secondary memory of a back-end were distributed randomly across the disk drives of that back-end. Since each track requires the time for seek and rotation we now consider that the tracks to be retrieved at a back-end are clustered in such a way that the seek time may be unnecessary for all except the first track retrieved. The number of back-ends is chosen from the set {5, 10, 15}. Another parameter that is varied is the number of tracks that must be retrieved for a typical request. A small-sized request requires the retrieval of between one and five tracks. A medium-sized request requires the retrieval of between

System / Inter Arrival Time of Requests (msecs)	NUMBER OF BACK-ENDS = 15		
	MDBS with Broadcast	MDBS without Broadcast	MDBS without Broadcast / MDBS with Broadcast
100	46.9	104	2.22
200	40.7	104	2.56

System / Inter Arrival Time of Requests (msecs)	NUMBER OF BACK-ENDS = 10		
	MDBS with Broadcast	MDBS without Broadcast	MDBS without Broadcast / MDBS with Broadcast
100	72.7	92.6	1.27
200	59.2	92.2	1.56

Table Va. Response Time for MDBS with and without Broadcast Facility

five and twenty tracks. Finally, for large-sized requests, the number of tracks to be retrieved varies from twenty to one hundred. The results for small-sized, medium-sized, and large-sized requests are shown in Table Vb, Vc and Vd, respectively.

Consider the results for the case where the number of tracks to be retrieved varies from one to five. With an interarrival time of one request every 200 msecs, the average response times for 5, 10 and 15 back-ends using broadcast is 31.7, 20.6 and 17.2 msecs, respectively. Thus, the response time decreases with increasing number of back-ends as expected. On the other hand, the corresponding response times for the case where no broadcast capability is assumed are 46.2, 59.3 and 87.0 msecs, respectively. That is, the response time actually increases with increasing number of back-ends! This is because the message sending time is dominating the secondary memory access time (since the number of tracks to be retrieved is so few). By comparing the set of response times with the broadcast capability with the corresponding set of response times without such a capability, we see that the use of broadcast may result in 50%, 130% and 400% of improvements for the respective configurations of 5, 10 and 15 back-ends. The results remain to be the same, even when the interarrival rate is increased to one request every 100 msecs.

Next, let us consider the results for the case where the requests are medium-sized and where the requests require the retrieval of between five and twenty tracks of data. Once again, the response time of MDBS without the broadcast capability increases with increasing number of back-ends. Thus, with an interarrival time of one request every 100 msecs and five back-ends, the response time is 85.1 msecs. However, the response time with ten back-ends is 87.4 msecs and with 15 back-ends it is 104 msecs. On the other hand, the system with the broadcast capability behaves in an increasingly better manner. The corresponding figures in this case are 96.4, 56.7 and 43.2. The percentage of improvement gets to be as high as 250% or 2.5 times better.

Finally, consider the results for the case where the number of tracks retrieved is between twenty and one hundred. The effect of not having a broadcast capability is expected to be felt the least under such circumstances. The simulation results certainly bear out this intuitively expected result. For the first time, the system without the broadcast capability shows an improvement in response time when the number of back-ends is increased from five to ten. However, the performance of the system does not improve when the number of back-ends

Response Time Tables (in milliseconds)

Small-Sized Requests

(Between 1 and 5 tracks)

Inter-arrival time = 100 msecs		
Number of back-ends	With Broadcast	Without Broadcast
5	34.5	46.3
10	21.8	59.3
15	18.1	86.9

Inter-arrival time = 200 msecs		
Number of back-ends	With Broadcast	Without Broadcast
5	31.7	46.2
10	20.6	59.3
15	17.2	87.0

Table Vb. Comparing MDBS with and without Broadcast for Small-Sized Requests

Response Time Tables (in msecs)

Medium-Sized Requests

(Between 5 and 20 tracks)

Inter-arrival time = 100 msecs		
Number of back-ends	With Broadcast	Without Broadcast
5	96.6	85.1
10	56.7	87.4
15	43.2	104

Inter-arrival time = 200 msecs		
Number of back-ends	With Broadcast	Without Broadcast
5	73.2	81.6
10	49.0	87.3
15	38.7	104

Table Vc. Comparing MDBS with and without Broadcast for Medium-Sized Requests

Response Time Tables (in msecs)
Large-Sized Requests
(Between 20 and 100 Tracks)

Inter-arrival time = 500 msecs		
Number of back-ends	With Broadcast	Without Broadcast
5	233	232
10	118	152
15	86.8	152

Table Vd. Comparing MDBS with and without Broadcast for Large-Sized Requests

is increased from ten to fifteen. In contrast, the performance of the system with the broadcast capability improves constantly with ever increasing number of back-ends. When the number of back-ends is 15, we note that the system with the broadcast capability still outperforms the system without the broadcast capability by 90%.

In conclusion, the use of a broadcast capability can be very important. The need becomes more acute when the number of back-ends is large and when typical requests do not require the retrieval of large amounts of data. A system without broadcast capability can behave anomalously in that the response time actually increases with increasing number of back-ends. Such degradation in response time is primarily due to the increase in control message traffic.

We wish to emphasize that the ability to broadcast is essentially attained by software means. That is, no special-purpose hardware is needed in order to achieve a broadcast capability. There are many examples of systems that provide a broadcast capability. For instance, Ethernet [Metc76], which is a coaxial cable network, provides such a capability. Another scheme which provides a broadcast capability is the time-shared bus. Three different commonly used techniques for achieving a broadcast capability using a time-shared bus are described in [Tane81] and will not be repeated here. In conclusion, broadcast capability is achieved by using appropriate software and requires no special-purpose hardware. Furthermore, such software is available from many vendors and is being commonly used. Hence, our proposal for using a broacast capability in MDBS does not lead to the problem of hardware specialization.

2.5 An Overview of the MDBS Architecture and Design

At this point, our proposed architecture looks as shown earlier in Figure 7. It consists of a single controller attached to a number of back-ends by way of a bus or a local network (such as an Ethernet) which provides a broadcast capability. A back-end is attached with a number of disk drives which may be accessed only by that back-end. Since we do not use special-purpose hardware, we have therefore solved the problem of hardware specialization in MDBS. In Chapter 1, we had listed eleven different issues which had to be studied in the context of a multiple back-end system. Let us briefly consider those issues that have been resolved for MDBS at this point of the design study.

The first issue was regarding the optimal way of interconnecting a large

number of back-ends and the optimal way of connecting the controller to the back-ends. We have decided on a scheme whereby all the back-ends and the controller are attached to a bus. This provides the controller with the ability to broadcast requests to all the back-ends and leads to a minimization of the control message traffic problem.

Another issue concerns the placement of data aggregates of the database across the back-ends. This issue has also been resolved to our satisfaction. We have shown, by simulation, a strategy with which the response sets to all requests may be evenly distributed among the back-ends and the extra tracks of data are assigned to consecutive back-ends starting from a randomly selected back-end. The strategy is best in terms of minimum response time and minimum storage wastage. We have termed it the track-splitting-with-random-placement strategy. This strategy thus eliminates the back-end limitation problem.

Another issue we had mentioned in Chapter 1 was the database store interconnection problem - that is, the attachment of disk drives to back-ends. We have chosen to adopt the dedicated approach because it eliminates the channel limitation problem. Furthermore, by attaching more than one disk drive to each back-end, we have eliminated the device limitation problem.

The next issue concerns the execution strategy. That is, should a SIMD or a MIMD approach be used. Our proposed design operates in a MIMD fashion. As much as possible, the different back-ends are made to work on the same request. However, when a back-end finishes the execution of a request, the back-end is to start the execution of the next request. In other words, the different back-ends are executing requests in an asynchronous fashion. Asynchronous execution is achieved in MDBS by having a queue of requests at each back-end. Whenever a back-end finishes a request, it picks up the next request from its queue and begins its execution. As we had indicated in our survey of DIRECT, this saves the need for a back-end to send a message to the controller after execution of each request and the need for the controller to respond with the next request in the controller queue. Thus, assuming a message time of eight milliseconds, this proposal will save sixteen milliseconds per request over a proposal where no queue of requests is maintained at each back-end. Additionally, it alleviates the controller limitation problem to some extent by removing the queue handling software from the controller and placing it at the back-ends. In fact, the amount of overhead associated with the controller is exactly the same as

would have been associated with it if the system had been executing in an SIMD mode. In other words, our proposal for a queue at each back-end allows us to reap the benefits of the MIMD mode of execution at the price of the SIMD mode of execution.

Thus, five of the nine design goals we set for ourselves in Section 2.2.1 have been achieved at this point. We will achieve the goal of finding a canonical data model in Chapter III. We will achieve the goal of using identical software in Chapter IV. The means for achieving the goal of eliminating the multiple request execution problem is described in Chapter VI. Finally, in order to achieve our goal of alleviating the controller limitation problem, the directory management, security enforcement and concurrency control algorithms are carefully designed. These algorithms are described in Chapters IV, V and VI, respectively.

3. THE CHOICE OF A DATA MODEL AND A DATA MANIPULATION LANGUAGE

In this chapter, we will develop a canonical data model for MDBS implementation. A canonical model must meet the following three criteria: the translation criterion, the partition criterion and the language criterion. These criteria will be elaborated in the following sections. An attribute-based model is proposed herein which is the best candidate to meet the criteria. With the canonical data model as a basis, we then propose a data manipulation language in which users may issue requests to MDBS. The language also encompasses the useful notion of a transaction. Finally, in this chapter, we contrast the basic requests with the aggregate requests. The latter perform aggregate operations such as summation, average, maximum and minimum, over attribute values.

3.1 Three Selection Criteria

We will discuss the criteria informally and briefly herein. A detailed discussion of the criteria will be included in the course of developing the canonical model, known as the attribute-based model.

3.1.1 The Translation Criterion

All the systems surveyed in Chapter II [Auer80, Dewi78, Miss80, Ston78] support the relational model of data. On the other hand, a large number of operational database systems [Datayy, Idmsyy, Systyy, Adabyy, Totayy] support either the hierarchical or the network data model. Therefore, there is the need to translate the existing hierarchical and network databases into relational databases before they may be used on relational systems. Such translation is a part of general studies of converting a database from one model to another and is known as database transformation [Bane80]. Furthermore, there is the need to translate the requests for the hierarchical or network database into requests for a relational database. Request translation is also a part of general studies of translating a data language to another and is known as query translation [Bane80]. To the best of our knowledge, complete solutions to the problems o database transformation and query translation among the aforementioned three models are not at hand. Hence, more research needs to be done before hierarchical and network databases may be supported entirely on a relational system. For this reason, we do not prefer the relational model. Similar reasons may be raised to reject the network and hierarchical data models. Database transformation and query translation constitute therefore the first criterion, i.e., the translation criterion, for

selecting a data model.

There are two more criteria that a data model must satisfy if we are to implement the data model in MDBS. These are referred to as the partition criterion and the language criterion. We will illustrate these two criteria by means of examples in the sequel. The examples will also illustrate why we choose to reject the network and hierarchical data models for MDBS implementation.

3.1.2 The Partition Criterion

Consider the sample network data model in Figure 10 where an inventory control database consists of four record types (namely, customer, order, item and part) and three set types (customer-order, order-item and part-item). The customer-order set links each customer to all his orders. Thus, Customer ABC is linked to orders 1, 2 and 3. Corresponding to each order, there can by any number of items. Thus, for example, items 1 and 2 correspond to order 1. Finally, a part in the inventory consists of a number of items. For example, part 2 consists of items 2, 3, 5, 7 and 8.

A typical request for such a database may be as follows:

'list all the order numbers for Customer ABC'.

A conventional network database system would respond to such a request by first retrieving the record for Customer ABC (with the use of a hashing function on the unique customer number) and then following the chain of pointers to make three additional secondary accesses to retrieve the records for orders 1, 2 and 3. Thus, a conventional system would need four secondary accesses to respond to this request.

Consider, now, how a system like MDBS would answer this request. For expository purposes, let us assume that MDBS consists of a controller and three back-ends. Now, the response time of MDBS to this particular request would depend upon the manner in which the database of Figure 10 is distributed across the three back-ends. If the record corresponding to Customer ABC, and those corresponding to orders 1, 2 and 3 are all stored at one of the back-ends, then MDBS, like the conventional system, would also need four accesses to the secondary memory. Clearly, we can do better than four.

One way to minimize the number of accesses might be to partition the database in such a way that the three records in the customer-order set with Customer ABC as their owner are placed in the three different back-ends. The

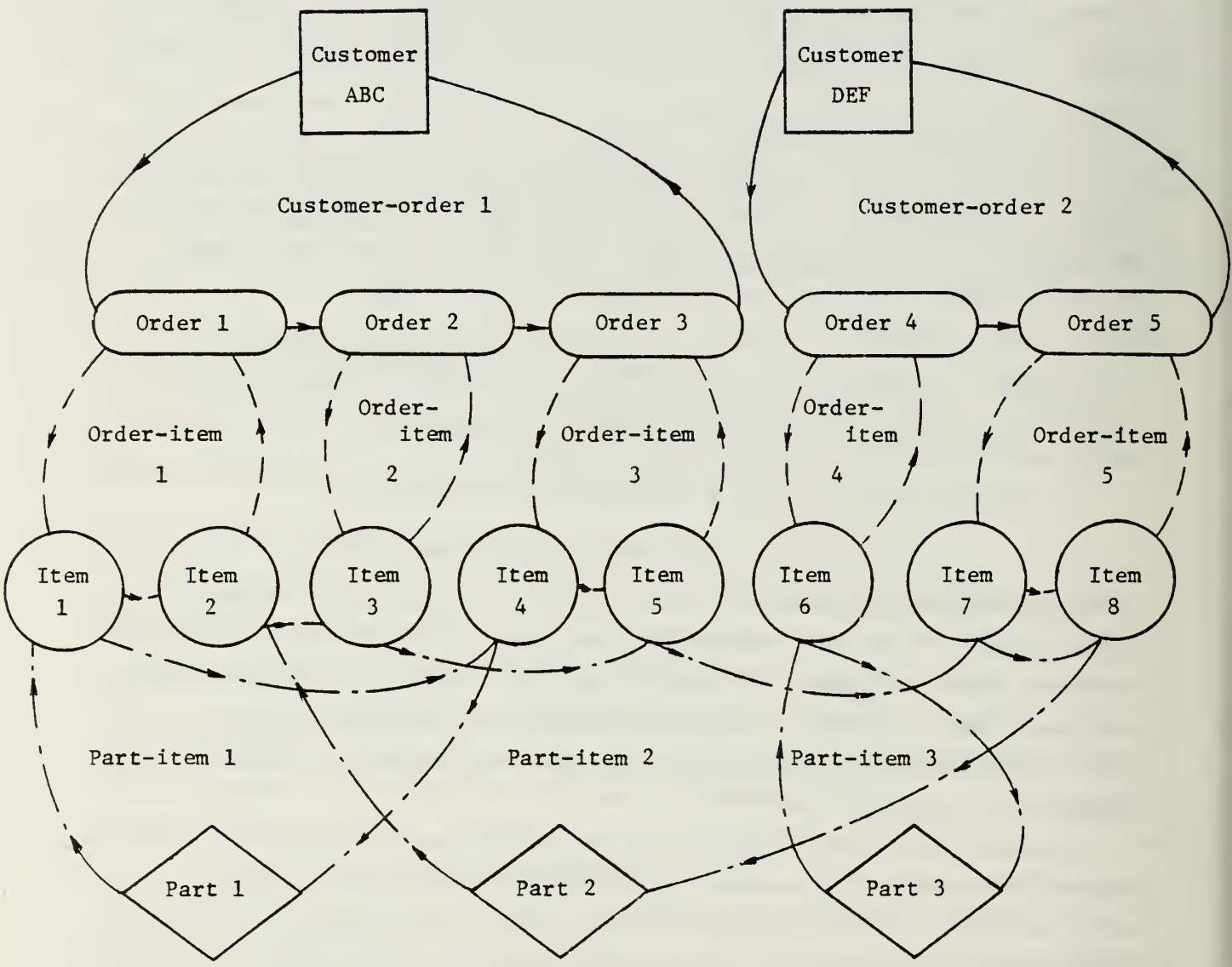


Figure 10. A Sample Network Database with Four Record Types and Three Set Types

new database storage is now shown in Figure 11. Essentially, we have partitioned the member records of the set customer-order by the back-ends. Thus, any request which requires the retrieval of all members in any customer-order set may be easily answered. The same request

'list all order numbers for Customer ABC'

is now answered in two accesses to the secondary memory, instead of the four needed in the previous environment.

We have, however, had to incorporate some redundancy in the database. Thus, the Customer ABC record is now to be stored in three different disks, the Customer DEF record is to be stored in two disks, the Part 1 record is to be stored in two disks and the Part 2 record is to be stored in three disks. The record storage increases by a factor of (24/18=) 1.5. Similarly, the number of pointers has gone up from 31 to 37.

In addition to the fact that the total amount of storage may go up is the new problem of updating redundant data. Thus, if the address of Customer ABC is changed, it must now be changed in three different back-ends. Hence, the controller must be aware of all the different copies of a record and must ensure the mutual consistency [Thom79] of all these different copies during update. Algorithms for ensuring the mutual consistency of these different copies are rather complicated. Thus, we should avoid the problem if possible.

Although the above solution to partitioning the network database has resulted in a faster response to requests for all members in any customer-order set, the response to requests for all members in an order-item set is as slow as in any conventional system. In order to improve the response time to such requests, we will need to partition the member records of all order-item sets by the back-ends. This causes a further increase in the amount of data redundancy.

Finally, we see that the insertion of a new record into the database will require the addition of other data. In referring to Figure 11, for example, if a new order is created for Customer DEF and must be inserted into the database, it should be inserted into back-end 3 in order to ensure an even distribution of the member records of Customer DEF across the back-ends. However, since back-end 3 does not contain a record for Customer DEF, a copy of the Customer DEF record must be created in back-end 3 in order to represent the relationship between Customer DEF and the newly inserted order record.

From the above discussion, we learn that the partitioning of a network

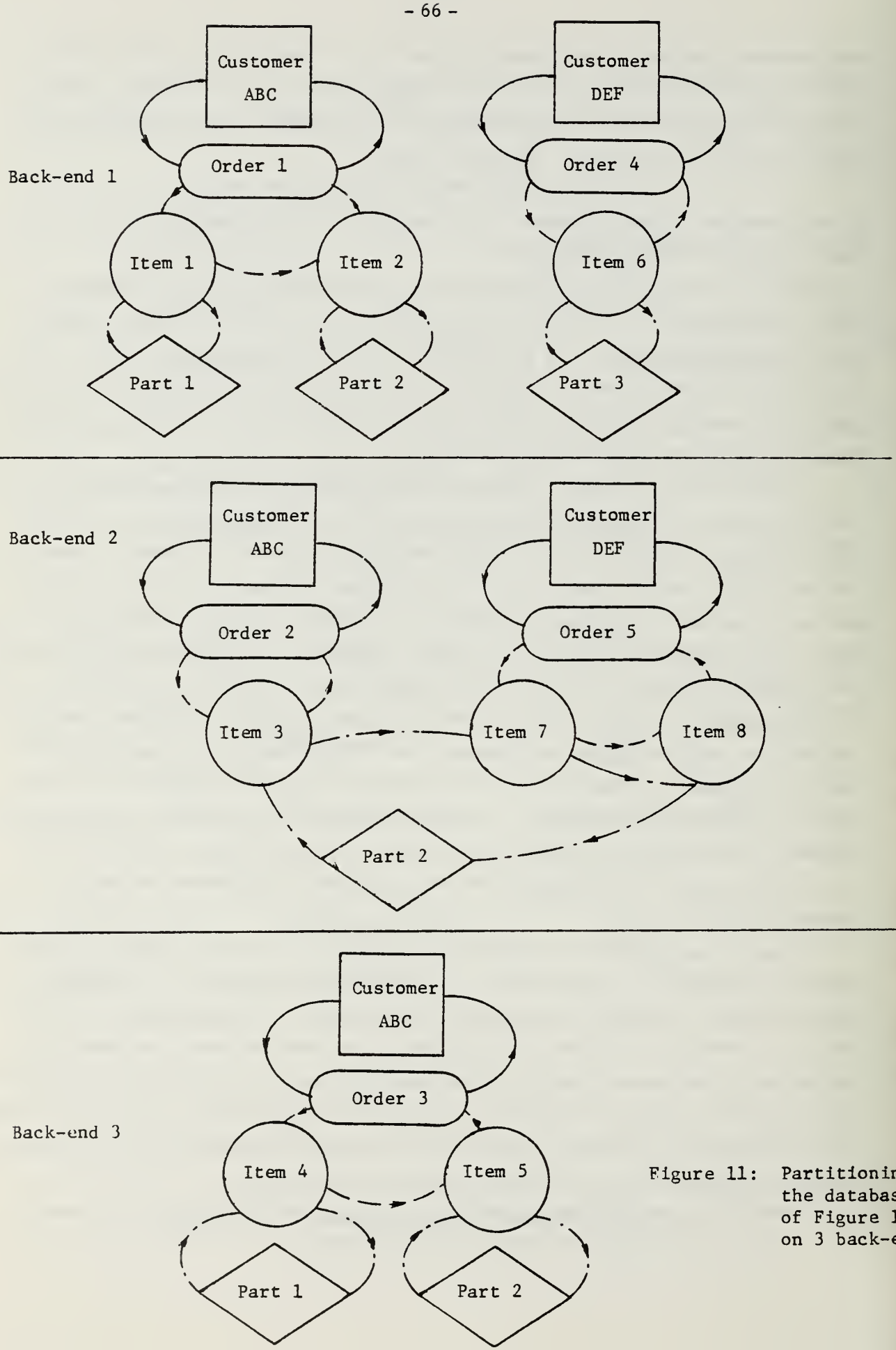


Figure 11: Partitioning the database of Figure 10 on 3 back-ends

database for MDBS may introduce the problems of storage redundancy and mutual consistency. A data model is easily partitionable if the partitioned database stored in MDBS for the model leads to little or no storage redundancy. Consequently, there will be little or no concern for the problem of mutual consistency during update. Both hierarchical and network models are not easily partitionable. On the other hand, the relational model is easily partitionable. This concludes our presentation of the partition criterion. It also illustrates why we are against implementing a network or hierarchical data model in MDBS.

3.1.3 The Language Criterion

Another reason for not implementing the hierarchical or network data model in MDBS is related to the data manipulation languages associated with these data models, i.e., the so called language criterion. It is evident that MDBS will outperform conventional systems for requests that require content-addressing and retrieval of large volumes of data, because the data can be spread across many back-ends and can be fetched in parallel. However, if the user transaction demands records in a sequential, one-record-at-a-time manner, then MDBS cannot outperform a conventional system to any great extent. Unfortunately, the data manipulation languages associated with hierarchical network databases tend to manipulate data in a sequential, one-record-at-a-time manner. Hence, they are unsuitable for MDBS implementation. The ideal language for MDBS will be one which is highly concurrent and which requires the retrieval of large volumes of data. We shall present such a language in a later section.

3.2 The Attribute-Based Model

Having eliminated from consideration the relational, hierarchical and network data models, we shall now consider the attribute-based data model [Hsia70, Roth80, Wong71]. In [Bane77, Bane78a, Bane80], several contributions are made with regard to the attribute-based model. First, they have shown that any relational, network or hierarchical database may be transformed, in a straightforward way, into an attribute-based database. Thus, there is no database transformation problem. They have also demonstrated that the requests issued in the data manipulation languages of these three data models may be easily translated into the requests of the attribute-based data manipulation language. Thus, the attribute-based data model does not suffer from the query translation

problem. Consequently, unlike the relational, hierarchical and network data models, the attribute-based model meets the translation criterion. Why is it that other data models and their manipulation languages can be so easily translated into the attribute-based model and its manipulation language? The reason has to do with the fact that the attribute-based model is a very basic model which embodies only a few simple concepts. When one tries to transform the database of a complex model into a database of another complex model, one must use the complex concepts in the second model to 'emulate' the complex concepts of the first model. This is difficult due mainly to the major differences among the concepts of these models. For example, the concept of a set of the network model is sufficiently complex as to make it difficult to find its counterpart in the relational and hierarchical models. In other words, it is not easy to find concepts in the relational and hierarchical models to emulate the concept of a set. On the other hand, being basic, the concepts of the attribute-based model may be used as building blocks for the more complex concepts of the aforementioned three data models. The process of translating a complex concept into one or more elementary concepts is frequently an easier task than the process of translating a complex concept to one or more complex ones.

Next, the attribute-based data model does not suffer from the partition problem. We recall that the network data model suffers from the partition problem because its partitioned database has a large amount of data redundancy. Such redundancy was essentially caused by the fact that two different mechanisms are used to represent data in a network database, where entities are represented by records and relationships are represented by pointers. In an attribute-based model, all logical concepts (i.e., entities and relationships) are represented by attribute-value pairs. Thus, data may be easily partitioned across the various back-ends with no redundancy. Finally, the data manipulation language of the attribute-based model does not operate in a sequential, record-at-a-time manner. By the use of boolean expressions of predicates as queries, it operates in a highly concurrent manner thus allowing us to utilize the capabilities of MDBS to the fullest. Accordingly, we choose to implement the attribute-based model, since it meets all three criteria, namely, the translation, partition and language criteria.

3.2.1 Concepts and Terminology

The smallest unit of data in MDBS is a keyword which is an attribute-value

pair, where the attribute may represent the type, quality, or characteristic of the value. Information is stored in and retrieved from MDBS in terms of records; a record is made up of a collection of keywords and a record body. The record body consists of a (possibly empty) string of characters which are not used for search purposes. For logical reasons, all the attributes in a record are required to be distinct. An example of a record is represented below:

(<File,EMP>, <Job,MGR>, <Dept,TOY>, <Salary,30000>).

The record consists of four keywords. The value of the attribute Dept, for instance, is TOY. In this dissertation, we will use "<", ">" to bracket a attribute-value pair; "(",")" to parenthisize a record; character strings with leading capitals for attributes; numerals or all capitals for values; and commas to separate the attribute-value pairs of a record.

MDBS recognizes several kinds of keywords: simple, security and directory. Simple keywords are intended for search and retrieval purposes. Security keywords are intended for access control and will be elaborated fully in Chapter V. Directory keywords are used for forming clusters. A cluster of records has a high probability of being retrieved from the back-ends together. Records of a cluster are therefore stored in close proximity. We will discuss the concept of a cluster and cluster algorithms in chapter IV. In this chapter, we present the attribute-based model, the concept of simple keyword and its data manipulation language.

A keyword predicate, or simply predicate, is of the form (attribute, relational operator, value). A relational operator can be one of $\{=,\neq,>,\geq,<,\leq\}$. A keyword K is said to satisfy a predicate T if the attribute of K is identical to the attribute in T and the relation specified by the relational operator of T holds between the value of K and the value in T. For example, the keyword <Salary,15000> satisfies the predicate (Salary>10000).

A descriptor can be one of three types:

Type-A: The descriptor is a conjunction of a less-than-or-equal-to predicate and a greater-than-or-equal-to predicate, such that the same attribute appears in both predicates. An example of a type-A descriptor is as follows:

$$((\text{Salary}\geq 2{,}000) \wedge (\text{Salary}\leq 10{,}000)).$$

More simply, this is written as follows:

$$(2{,}000\leq \text{Salary}\leq 10{,}000).$$

Thus, for creating a type-A descriptor, the database creator merely specifies an attribute (i.e., Salary) and a range of values ($2,000 and $10,000) for that attribute. We term the value to the left of the attribute the lower limit and the value to the right of the attribute the upper limit.

Type-B: The descriptor is an equality predicate. An example of a type-B descriptor is:

(Position=PROFESSOR).

Type-C: The descriptor consists of only an attribute name, known as the type-C attribute. Let us assume that there are n different keywords K_1, K_2, ..., and K_n in the records of a database with a type-C attribute. Then, the type-C descriptor is really equivalent to n type-B descriptors B_1, B_2, ... and B_n, where B_i is the equality predicate satisfied by K_i. In fact, this type-C descriptor will cause n different type-B descriptors to be formed. From now on, we shall refer to the type-B descriptors formed from a type-C descriptor as type-C sub-descriptors.

For instance, consider that Dept is specified as a type-C attribute for a file of employee records. Furthermore, let all employees in the file belong to either the TOY department or the SALES department. Then two type-B descriptors will be formed for this file. They are (Dept=TOY) and (Dept=SALES).

In forming descriptors, the database creator must observe certain rules as follows:

(1) Ranges specified in type-A descriptors for a given attribute must be mutually exclusive.

(2) For every type-B descriptor of the form (attribute-1 = value 1), no type-A descriptor can have the same attribute (i.e., attribute-1) and a range that contains the same value (i.e., value-1).

(3) An attribute that appears in a type-C descriptor must not also appear in a type-A or a type-B descriptor defined previously.

(4) Type-A descriptors are specified first; Type-B descriptors next; Type-C descriptors last.

A keyword is said to be derived or derivable from a descriptor if one of the following holds:

(a) The attribute of the keyword is specified in a type-A descriptor and the value is within the range of the descriptor.

(b) The attribute and value of the keyword match those specified in a type-B descriptor.

(c) The attribute of the keyword is specified in a type-C descriptor.

The use of these descriptors will be demonstrated in Chapter 4.

A query conjunction, or simply conjunction, is a conjunction of predicates. An example of a query conjunction is:

(Salary>25000) ∧ (Dept=TOY) ∧ (Name=JAI)

We say that a record satisfies a query conjunction if the record contains keywords that satisfy every predicate in the conjunction.

A query is a boolean expression of predicates. An example of a query is:

((Dept=TOY) ∧ (Salary<10000)) ∨ ((Dept=BOOK) ∧ (Salary>50000))

3.2.2 The Data Manipulation Language (DML)

The data manipulation language for MDBS is a non-procedural language which supports four different types of requests - retrieve, insert, delete and update. The syntax of these various requests and examples of them are presented below. For a formal specification of DML, the reader may refer to the Appendix A.

A. Retrieve

The syntax of a retrieve request is:

RETRIEVE Query Target-list [BY Attribute][WITH Pointer]

That is, it consists of five parts. The first part is the name of the request. The second part is the query (as defined in Section 3.2) which characterizes the portion of the database to be retrieved. The target-list is a list of elements. Each element is either an attribute, e.g., Salary, or an aggregate operator to be performed on an attribute, e.g., AVG(Salary). We will support five aggregate operators - AVG, SUM, COUNT, MAX, MIN - in MDBS. An example of a target-list of two elements is (Dept, AVG(Salary)). The values of an attribute in the target-list are retrieved from all the records identified by the query. If no aggregate operator is specified on the attribute in the target-list, its values in all the records identified by the query are returned directly to the user or user program. If an aggregate operator is specified on the attribute in the target-list, some computation is to be performed on all the attribute values in the records identified by the query and a single aggregate value is returned to the user or user program. The fourth part of the request, referred to as the BY-clause, is optional as designated by the square brackets around it. The use of the By-clause is explained by means of an example. Assume that employee records are to divided into groups on the basis of the departments for the purpose of calculating the average

salary for all the employees in a department. This may be achieved by using a retrieve request with the specific target list, (AVG(Salary)), and the specific BY-clause, BY Department. Finally, the fifth part of the request, which is also optional, is a WITH-clause which specifies whether pointers to the retrieved records must be returned to the user or user program for later use in an update request. Some examples of retrieve requests are presented below.

Example 1. Retrieve the names of all employees who work in the Toy Department.

RETRIEVE (File=EMPLOYEE) ∧ (Dept=TOY) (Name)

Example 2. Retrieve the names and salaries of all employees making more than $5000 per year.

RETRIEVE (File=EMPLOYEE) ∧ (Salary>5000) (Name,Salary)

Example 3. Find the average salary of an employee.

RETRIEVE (File=EMPLOYEE) (AVG(Salary))

Example 4. List the average salary of all departments.

RETRIEVE (File=EMPLOYEE) (AVG(Salary)) BY Department

B. Insert

The syntax of an insert request is:

INSERT Record

where, Record is the record to be inserted into the database. An example of an insert command is:

INSERT (<Relation,EMPLOYEE>,<Salary,5000>,<Dept,TOY>)

C. Delete

The syntax of a delete request is:

DELETE Query

where Query is a query which specifies the particular records to be deleted from the database. An example of a DELETE request is:

DELETE (Name=HSIAO) ∨ (Salary>50)

D. Update

In DML, the syntax of an update request is:

UPDATE Query Modifier

where Query specifies the particular records from the database to be updated and Modifier specifies the kinds of modification that need to be done on re-

cords that satisfy the query. An update in MDBS which provides for the modification of only a single attribute value will have the attribute referred to as the attribute being modified. The modifier in an update request specifies the new value to be taken by the attribute being modified. The new value to be taken by the attribute being modified is specified as a function f of the old value of either that attribute (i.e., Type-I) or some other attribute (Types II, III and IV). The modifier may be one of the following five types:

Type-0: <attribute=constant>

Type-I: <attribute=f(attribute)>

Type-II: <attribute=f(attributel)>

Type-III: <attribute=f(attributel) of Query>

Type-IV: <attribute=f(attributel) of Pointer>

Let a record being modified be referred to as the record being modified. Then, a Type-0 modifier sets the new value of the attribute being modified to a constant. A Type-I modifier sets the new value of the attribute being modified to be some function of its old value in the record being modified. A Type-II modifier sets the new value of the attribute being modified to be some function of some other attribute value in the record being modified. A Type-III modifier sets the new value of the attribute being modified to be some function of other attribute value in another record uniquely identified by the query in the modifier. Finally, a Type-IV modifier sets the new value of the attribute being modified to be some function of some other attribute value in another record identified by the pointer in the modifier. An example of a Type-0 modifier is:

<Salary=5000>

This sets the salary of all the records being updated to 5000.

An example of a Type-1 modifier is:

<Salary=1.1xSalary>

This raises the salary of all qualifying employees by 10%.

An example of a Type-II modifier is:

<Monthsal=Yearsal/12>

This sets the monthly salary of all qualifying employees to be a twelfth of their own yearly salaries.

An example of a Type-III modifier is:

<Salary=Salary of (Relation=WIFE) ∧ (Name=TARA)>.

An example of a Type-IV modifier is:

<Salary=Salary of 2000>

which modifies the salary of all qualifying employees to that of the record stored in location 2000. In order to use this type of modifier, the user must have previously issued a retrieve request with the WITH POINTER option. An example of a complete update request may be:

UPDATE (File=EMPLOYEE) <Salary=Salary+25>

which gives a raise of $25 to all employees.

3.2.3 The Notion of a Transaction

In DML, we allow the flexibility for a user to specify a set of requests for repeated execution. Such a pre-specified set of requests shall be referred to as a transaction. As in other systems, a transaction must preserve consistency, i.e., uphold the truthhood of the assertions made about the database. A database creator specifies a set of assertions on the database. These assertions are constraints which must be satisfied by data in the database. For instance, since employees cannot have negative salaries, an assertion on the database may require that all employees have positive salaries. An assertion about a database is said to be true in the database if the data in the database satisfies the constraints in the assertion. A database is in a consistent state if all the assertions made on the database by the database creator remain to be true in the database. Finally, a transaction is said to preserve consistency, if the database is in a consistent state before it begins execution and the database is in a consistent state after it finishes execution.

Some examples of transactions in our environment are present below. We begin each transaction with BOT, an acronym for beginning-of-transaction and we terminate each transaction with EOT for end-of-transaction.

Example 1. We assume the existence of two files in a database [Eswa76], one called accounts and the other called assets depicted in Figure 12. The only assertion made by the database creator on this database is that the total assets at a particular location are equal to the sum of the balances at that location.

We Assume that the record

(<Location,NAPA>, <Number,5320>, <Balance,287>)

is to be inserted repeatedly into the database. Each time such an insertion takes place, the sum of the balances at location Napa is increased by $287. Hence, if the assertion, that the sum of the balances at a location is equal to the total assets

ACCOUNTS		
Location	Number	Balance
NAPA	32123	1050
ST HELENA	36592	506
NAPA	5320	287

ASSETS	
Location	Total
ST HELENA	506
NAPA	1337

Figure 12. A Sample Database of Two Files
(Adopted from [Eswa76])

at that location, is to be preserved, then the total for the location Napa in the assets file must also be increased by $287. A transaction that does this is:

```
BOT
INSERT (<Location,NAPA>, <Number,5320>, <Balance,287>)
UPDATE (File=ASSETS) ∧ (Location=NAPA) <Total=Total+287>
EOT
```

Example 2: From time to time, employee Smith is given a raise of $50. An assertion on the database may require that employee Jones always makes the same salary as employee Smith. Hence, the salary of Jones must also be raised by $50 whenever the salary of Smith is raised by $50. A transaction that does this is:

```
BOT
UPDATE (File=EMPLOYEE) ∧ (Name=SMITH) <Salary=Salary+50>
UPDATE (File=EMPLOYEE) ∧ (Name=JONES) <Salary=Salary+50>
EOT
```

3.2.4 Basic Requests vs. Aggregate Requests

We have thus far described the various basic types of requests of MDBS. Detailed algorithms for the execution of the basic request types will be presented in Chapter 4, although, in Chapter 2 we had briefly described the execution sequence of the retrieve requests in MDBS. There, we had stated that a retrieve request would be initially broadcast from the controller to all the back-ends over a bus and that the results would be output by each back-end to the controller via the same bus. We will elaborate on the performance and the execution sequences of all the basic types of requests in Chapter 4.

When an aggregate operator is included in a request, the execution of the request takes additional complexity, since considerable computations must be performed by MDBS on data aggregates for the request. For this reason, we study requests employing aggregate operators. From now on, we will refer to requests using aggregate operators as aggregate requests. Furthermore, we will analyze the performance of MDBS in the execution of aggregate requests in Chapter 4.

3.2.5 In Meeting the Selection Criteria

In the beginning of this chapter, we stated three criteria which were to be satisfied by a data model and the data manipulation language of the data model.

The three criteria were intended for the selection of an ideal data model and its data manipulation language for MDBS implementation. These criteria were referred to as the translation criterion, the partition criterion and the language criterion.

The attribute-based model satisfies the selection criteria as we amply argued in the preceeding sections. In this section, we consider its data manipulation language, DML, in terms of the selection criteria.

Of the three criteria, the partition criterion is to be met by the model alone. Hence, we do not need to consider this criterion for DML. We shall consider the remaining criteria for DML in the following paragraphs.

In [Bane80], it was demonstrated that the data manipulation languages of the three prevailing data models - network, hierarchical and relational - could be translated into the data manipulation language of a database computer, known as DBC. The data manipulation language proposed in this chapter, i.e., DML, properly contains all the features of the data manipulation language of DBC. Hence, if the data manipulation languages of the three prevailing data models can be translated into the data manipulation language of DBC, they can also be translated into DML. Thus, DML satisfies the translation criterion.

Finally, DML allows for the concurrent access of large volumes of data because it enables the user to specify queries in terms of boolean expression of predicates. Consequently, DML satisfies the language criterion.

In conclusion, the proposed DML satisfies all the criteria indicated in the beginning of this chapter.

4. THE PROCESS OF REQUEST EXECUTION

In Chapter 3, we had described the data manipulation language, DML, used in MDBS. In this Chapter, we will describe the process of DML request execution from the time a DML request is first received by MDBS to the time the response set of the request is returned to the user or user program.

Central to the discussion on request execution is the notion of a cluster. This notion will be developed in Section 4.1. Here, we give a brief introduction to the process of request execution. We first utilize the descriptors defined in Chapter 3 as an equivalence relation to partition the database into equivalence classes which are termed clusters. The equivalence relation guarantees the following nice properties: Every record in the database belongs to one and only one cluster of the database. By proper use of the descriptors, the clusters may be formed in such a way that if a user needs to access a record belonging to a cluster, the user is most likely to have the need to access all the other records belonging to that cluster. Thus, clusters serve as the basic units of access in MDBS and every database is stored in MDBS as a collection of clusters.

The execution of a user request in MDBS proceeds typically in several distinct phases as follows. In the first phase, MDBS will determine the exact clusters of records which will satisfy the request. In the second phase, MDBS accesses security information about the user in order to select for the user the authorized clusters among the clusters which have been determined in the first phase. In the third phase, MDBS determines the secondary memory addresses of the authorized clusters selected in the second phase. In all these three phases, MDBS makes no access to the records of the clusters selected. Instead, MDBS utilizes auxiliary information about the clusters and security. Such utilization of auxiliary information constitutes the directory management function of MDBS. After the three phases of directory management, MDBS retrieves the clusters of records which will satisfy the request.

The use of the data placement strategy in MDBS will ensure us that each cluster (database) is stored in such a way that the records (clusters) of the cluster (database) are evenly distributed across the multiple back-ends of MDBS. Since all clusters are evenly distributed across the back-ends, the response set of the request will be retrieved in a parallel fashion from the back-ends.

This completes our brief description of the processing of a request in MDBS. The remaining sections of this Chapter are organized as follows. In Section 4.1, we will develop the notion of a cluster. The data structures to be used for deter-

mining the set of clusters which will satisfy a request, and the algorithms necessary for such determination are also described in Section 4.1. In keeping with our findings in Chapter 2, each cluster is placed according to the track-splitting-with-random placement policy which was shown to be superior. In Section 4.2, we will describe two of the three phases of the directory management. Phase 2 is dealing with security-related directory management. Because of its importance and our new contribution, it is discussed in Chapter 5. Several strategies for performing directory management are proposed and evaluated on the basis of a criteria which strives for minimal processing time. These same strategies are also evaluated, in Section 4.2, in terms of storage requirements. At the end of Section 4.2, we will present our recommendations for a directory management strategy for MDBS. In Section 4.3, we will describe the execution sequences for the various DML requests that may be issued to MDBS.

4.1 The Notion of Record Clusters

Record clusters are formed for the purposes of narrowing the search space and minimizing the effort needed to retrieve records which may satisfy a given request. In other words, by organizing a database into clusters and by maintaining information about these clusters, MDBS may readily identify those clusters whose records will satisfy the given request, thereby achieving high throughput and good response time.

Although the notion of a record cluster for the above-mentioned purposes is well known, the effectiveness of clusters for throughput gain and response-time improvement lies in the effectiveness of the clustering algorithm for forming clusters and, more importantly, the placement strategy for storing these clusters. In other words, it depends on how clusters are formed and placed. Interestingly enough, it does not depend on how clusters are used. In other words, the throughput and response time of MDBS are 'immune' from the way the clusters are utilized. This is because every request execution by MDBS will involve the search and re-

trieval of clusters. Such search and retrieval can always be shown to be maximal for throughput gain and response-time improvement. We will comment on this point briefly herein and elaborate on the point thoroughly in the later sections. Briefly, this is due to our use of the descriptors as a means to define and form clusters. As we recall from Chapter 3, a descriptor is either a single predicate or a conjunction of predicates. We may also recall that a query in a user request is a boolean expression of predicates. Thus, a given user request will require the retrieval of data which satisfy the predicates of the expression. Since clusters are formed by the definition of descriptors and both descriptors and queries utilize the common notion of predicates, the data retrieved for the request are actually one or more clusters. Clusters therefore become the ideal formation (or unit) of data for storage and retrieval and for performance optimization.

Our work on clusters is unique also in a couple of respects. First, the clustering algorithms in our method will be executed in multiple back-ends rather than at a single back-end as in all other work. Second, we recongnize the importance which must be given to the placement of these clusters across the multiple back-ends of MDBS.

In the following sections, we will describe how the clusters are formed in MDBS and how they are used. We will begin with some definitions.

4.1.1 Cluster Formation

For a database, the creator of the database specifies a number of descriptors called clustering descriptors, or simply, descriptors. An attribute that appears in a descriptor is called a directory attribute. We say that a directory attribute belongs to a descriptor if the attribute appears in that descriptor.

We recall that a record consists of attribute-value pairs or keywords. For purposes of clustering, only those keywords of the record which contain directory attributes are considered. Such keywords of the record are termed directory keywords. From the rules for forming descriptors specified in Chapter 3, it is easy to see that a directory keyword is derivable from at most one descriptor. For example, consider a database with Salary as the only directory attribute. Furthermore, let (0≤Salary≤500) be the only descriptor D1 on Salary specified by the database creator. Now, consider two records, one containing the directory

keyword <Salary,250> and the other containing the directory keyword <Salary,750>. Clearly, the former directory keyword is derivable from descriptor D1 and the latter directory keyword is not derivable from D1. Hence, the latter keyword is not derivable from any descriptor in the database and we say that the directory keyword is derivable from no descriptor. Since a record may have many directory keywords, each of which will be derivable from at most one descriptor, we say that the record is derived from a set of descriptors. It is possible for a record to be derived from the empty set of descriptors. There are two such cases. In the first case, it may happen that a record does not contain any directory keyword. In this case, it is said that the record is derived from the empty set of descriptors. Thus, going back to the previous example with the single directory attribute, Salary, and the single descriptor, (0≤Salary≤500), a record which does not contain any salary information (i.e., no keyword with the attribute Salary) is said to be derived from the empty set of descriptors. The second case in which a record is derived from the empty set of descriptors is when the record does indeed contain directory keywords, but these keywords are not derivable from any descriptors. In the previous example, a record with the directory keyword <Salary,750> which is not derivable from the descriptor is therefore derived from the empty set of descriptors also.

If two records are derived from the same set of descriptors, they are likely to be retrieved together in response to a user request, since these two records have keywords which are derived from the same set of descriptors. Thus, these two records should be stored together in the same cluster. A cluster is, therefore, a group of records such that every record in the cluster is derived from the same set of descriptors. We say that a record cluster is defined by the set of descriptors from which all records in the cluster are derived.

It is easy to see that a record belongs to one and only one cluster. The reasoning is as follows. A record consists of zero or more directory keywords. If it consists of zero directory keywords, it belongs to the cluster defined by the empty set of descriptors. If the record consists of one or more directory keywords, then, the record must be derived from one and only one set of descriptors, since each directory keyword is derived from at most one descriptor. This unique set of descriptors defines the unique cluster to which the record belongs. Thus, we have used the concept of descriptor sets to partition the database into equivalence classes, namely clusters. A formal proof of the above observations is included in an Appendix B.

In order to form clusters for the records in a database, an algorithm is

provided herein which will take a record and determine its cluster. We will describe this algorithm informally below. The detailed algorithm to be implemented in MDBS for cluster formation will be presented in Appendix C.

The algorithm for determining the cluster to which a record belongs is as follows. For each attribute-value pair in the record, determine if the attribute is a directory attribute. If it is not, then that attribute-value pair is not used for cluster determination. If the attribute is a directory attribute, determine the descriptor, if any, from which it is derived. We refer to this descriptor, if any, as the corresponding descriptor for the given attribute-value pair. The set of corresponding descriptors for all the attribute-value pairs in a record defines the cluster to which the record belongs.

By using the algorithm on every record of a database at database-creation time, we may form the record clusters of the database.

4.1.2 An Example of Cluster Formation

Consider the database and its descriptors shown in Figure 13. The figure shows two files, accounts and assets. The accounts file has three records and the assets file has two records. Also, the figure shows the five descriptors specified on this database by the database-creator. The attributes, Number and Balance, are type-A descriptors and the attribute, Location, is a type-C descriptor. This type-C descriptor will be converted into two type-B descriptors as shown in Figure 14. There is no descriptor for the attribute, Total, because no request with the attribute, Total, as part of a query is expected. The clusters formed for this database and the records in each cluster are depicted in Figure 15.

The first row in Figure 15 is for cluster 1. The set of descriptors defining this cluster is {D2, D3, D5}. Consider record R1 which belongs to this cluster. It contains the keyword <Location, NAPA> which is derived from the descriptor D5 (Location=NAPA). Similarly, the keyword <Number,32123> is derived from D2 which is (15000≤Number<∞). Finally, the keyword <Balance,50> is derived from D3 which is (0≤Balance<500). Hence, R1 belongs to the cluster defined by D2, D3 and D5. Similarly, it may be shown that R3 belongs to the same cluster. This explains the first row in Figure 15. With similar exercises, we may verify the remaining three rows in Figure 15.

Accounts file:

(<Location,NAPA>, <Number,32123>, <Balance,50>)
(<Location,St. Helena>, <Number,5320>, <Balance,506>)
(<Location,NAPA>, <Number,36592>, <Balance,287>)

Assets file:

(<Location,St. Helena>, <Total,506>)
(<Location,NAPA>, <Total,337>)

The database creator specifies the following descriptors:

Type-A Descriptors:

$(0 \leq \text{Number} < 15000)$
$(15{,}000 \leq \text{Number} < \infty)$
$(0 \leq \text{Balance} < 500)$
$(500 \leq \text{Balance} < 1000)$

Type-C Descriptor: Location

* For simplicity, we refer to the three records of the accounts file as R1, R2 and R3, respectively; and to the two records of the assets file as R4 and R5, respectively.

Figure 13. A Database of Two Files and its Clustering Descriptors

Descriptors Specified by the Database-Creator as seen by MDBS

Descriptor	Descriptor id
(0 ≤ Number < 15,000)	D1
(15,000 ≤ Number < ∝)	D2
(0 ≤ Balance < 500)	D3
(500 ≤ Balance < 1000)	D4
(Location = Napa)	D5
(Location = St. Helena)	D6

The Descriptors Formed for the Database of Figure 13

Figure 14. The Descriptor-to-Descriptor-Id Table (DDIT)

The Clusters Formed for the Database of Figure 13

Cluster	Set of Descriptors*	Records in the Cluster Defined by the Descriptor Set**
1	{D2, D3, D5}	R1, R3
2	{D1, D4, D6}	R2
3	{D6}	R4
4	{D5}	R5

* Actually, only descriptor ids are shown here, Together with the Descriptor-To-Descriptor-Id Table shown in Figure 14, MDBS maintains the descriptor sets.

** In implementation, this column contains the secondary memory addresses of R1, R2, R3, R4 and R5 with two addresses in the first row.

Figure 15. The Cluster-Definition Table (CDT)

4.1.3 Clusters Determination During Request Execution

Up to this point, we have been describing the process of cluster formation. We will now explain how clusters are used during request execution. More specifically, we will explain how to determine the cluster to which a new record belongs and how to determine the set of clusters which must be retrieved in order to satisfy a query for retrieval, deletion or update.

During the process of cluster formation described in the previous section, MDBS uses an algorithm repeatedly for determining the cluster of a record in the database. This same algorithm may now be used by MDBS to determine the cluster of a record for the record insertion. In insertion, the cluster definition table (CDT) is used in order to determine the secondary memory address (addresses) of this cluster.

Next, let us describe how MDBS determines the set of clusters which satisfy the query in a retrieval, deletion or update request. Before we may do this, we must introduce some concepts and terminology.

Descriptor X is defined to be less than descriptor Y, if the attributes in both descriptors are the same and one of the following holds.

(1) Both descriptors are of type-A and the upper limit of descriptor X is lower than the lower limit of descriptor Y.

(2) Both descriptors are of type-B and the value in descriptor X is smaller than the value in descriptor Y.

(3) Descriptor X is of type-A and descriptor Y is of type-B and the upper limit of descriptor X is lower than the value in descriptor Y.

(4) Descriptor X is of type-B and descriptor Y is of type-A and the value in descriptor X is smaller than the lower limit of descriptor Y.

An exactly parallel description for the greater-than relation among descriptors may also be given. The above definition covers the case where either X or Y is a type-C descriptor, since type-C descriptors are stored as type-B descriptors in MDBS.

To illustrate the definition of less-than among descriptors, let us assume that we are given the descriptors D1 (100≤Salary<200), D2 (0≤Salary<99), D3 (Salary=99) and D4 (Salary=200). Thus, D3 is less than D1; D2 is less than D3; and D1 is less than D4. The relation, less-than, is transitive. Hence, we can define a partial ordering among descriptors. For example, the ordering among these four descriptors is D2, D3, D1 and D4.

Using the above definition of less-than and greater-than for the descriptors,

we are ready to describe the algorithm for determining the corresponding set of clusters for a query in a user request. The query is assumed to be in disjunctive normal form, i.e., disjunction of conjunctions. The algorithm will proceed in three steps.

Since a query conjunction consists of predicates, we will determine, in the first step, a corresponding descriptor or a corresponding set of descriptors for each predicate. This is done as follows. If the predicate in a query conjunction is an equality predicate, then the corresponding descriptor is the one from which the keyword satisfying the predicate is derived. For example, if the predicate is Location=NAPA, then the keyword satisfying the predicate is <Location,NAPA> and the corresponding descriptor is (Location=NAPA). If the predicate is either a less-than or less-than-or-equal-to predicate, it is first treated as an equality predicate and the corresponding descriptor D for that equality predicate is first determined. Then, all the descriptors less than D, along with D, form the corresponding set of descriptors for the less-than or less-than-or-equal-to predicate. If the predicate is a greater-than or greater-than-or-equal-to predicate, then it is first treated as an equality predicate and the corresponding descriptor D for that equality predicate is first determined. Then, all the descriptors greater than D, along with D, form the corresponding set of descriptors for the greater-than or greater-than-or-equal-to predicate. Thus, we have determined a corresponding set of descriptors for a predicate.

The above procedure is repeated for every predicate in the query conjunction. Thus, we will have determined a corresponding set of descriptors for every predicate in a query conjunction.

Our next step is to determine the corresponding set of clusters for a query conjunction, since a query consists of one or more query conjunctions. Let the query conjunction have p predicates. Let the set of descriptors corresponding to the i-th predicate be S_i. Now, form all possible groups, where each group consists of one descriptor from S_i for i ranging from 1 to p. In other words, we are forming the cross-product of S_i. The reason for forming this cross-product of p sets is because a query conjunction consists of a conjunction of p predicates, each of which has a corresponding set S_i of descriptors. Each element in this cross-product is termed a descriptor group which is of course a set of descriptors. Intuitively, a group defines a set of clusters whose records satisfy the query conjunction.

We recall that MDBS maintains a table, known as the cluster definition table,

which is created at the database creation time. (See Figure 15 again for an example). However, the definitions kept in the table may not be identical to the definitions of the groups. Without relating the descriptor groups with the descriptor sets kept in the table, we may not be able to determine the clusters involved. Thus, this second step includes the determination of whether there are descriptor sets in the table which contain a descriptor group. If there are such sets, then the clusters defined by the descriptor sets are indeed the clusters referred to by the descriptor group.

By repeating this procedure for every descriptor group in the cross-product, we are able to determine the corresponding set of clusters for a query conjunction. The entire second step which is used to determine the corresponding set of clusters for a query conjunction is then repeated for every query conjunction in the query. Thus, we have determined a corresponding set of clusters for every query conjunction in the query.

The final step of the algorithm determines the corresponding set of clusters for the query from the corresponding set of clusters for each query conjunction in the query. Since the query is a disjunction of conjunctions, the corresponding set can be simply obtained as the union of the sets of clusters for each query conjunction in the query.

The three steps involved in this algorithm are illustrated with an example in the following section and formally specified in Appendix C.

4.1.4 An Example of Clusters Determination During Request Execution

Our example is developed around the database and descriptors of Figures 13 and 14. Consider that the following retrieve request is received by MDBS.

RETRIEVE ((Location=St.Helena) ∧ (Balance=506)) ∨ (Number<5500)(Balance)

Clearly, the corresponding descriptor for the predicate (Location=St. Helena) is D6 and that for the predicate (Balance=506) is D4. Similarly, the corresponding descriptor for the predicate (Number<5500) is D1. We complete the first step of the algorithm for determining the corresponding set of descriptors for each predicate.

In the next step, we need to determine the corresponding set of clusters for each query conjunction. Consider the first query conjunction, (Location= St. Helena) ∧ (Balance=506). The only descriptor group that can be formed for this query conjunction is {D6, D4}. In searching the entries of the descriptor

definition table depicted in Figure 15, we discover that there is only one descriptor set which contains the descriptor group. It is the set {D1, D4, D6}. The cluster defined by {D1, D4, D6}, i.e., cluster 2, is the only member of the corresponding set of clusters for the first query conjunction. Similarly, we determine the corresponding set of clusters for the second query conjunction, (Number<5500). It so happens that cluster 2 is also the only member of the corresponding set of clusters for the conjunction. Thus, we complete the second step of the algorithm.

In the final step of the algorithm, the union of all members of the corresponding sets of clusters of the query conjunctions is still cluster 2. Thus, cluster 2 constitutes the only member of the corresponding set of clusters for the given query. Once the corresponding set of clusters is determined, the addresses of the records in the clusters may be used for access to the secondary memory.

4.2 Directory Management

The entire sequence of actions taken by MDBS from the time a record-insertion request is received to the time that the cluster to which the record is to be inserted is determined (i.e., the secondary memory address or addresses are generated) is referred to as directory management for an insert request. Similarly, the entire sequence of actions taken by MDBS from the time a retrieve, delete or update request is received to the time the corresponding set of clusters for the query in the request and their addresses in the secondary memory are generated is referred to as directory management for a non-insert request. Together, they constitute the directory management of MDBS.

We repeat that the directory management in MDBS consists of three major phases. In the first phase, MDBS determines the exact clusters of records which will satisfy the user request. This phase was described in the previous section. The algorithm used in this phase for determining the clusters was briefly described in the previous section and in detail in an appendix. In the second phase, MDBS accesses security information about the user in order to select for the user the authorized clusters among the clusters which have been

determined in the first phase. This discussion is relegated to Chapter 5. In the third phase, MDBS determines the secondary memory addresses of the authorized clusters selected in the second phase.

4.2.1 Two phases of Processing - Descriptor Processing and Address Generation

For implementation of the directory management function, we recall that the cluster definition table CDT is used to determine the corresponding set of clusters for a query. At the same time, the secondary memory addresses of the corresponding set of clusters may also be found,since these addresses are present in the third column of CDT (see again Figure 15). However, we want to ensure that only the secondary memory addresses of the authorized clusters for a user are utilized. This is achieved by augmenting CDT of Figure 15 with several more columns of security-related information, one column for each user of the database. The details of the kinds of security-related information maintained for each user are given in Chapter 5. Thus, in implementation of the directory management function, the three phases described above may be elaborated as follows: In the first phase, the descriptor-to-descriptor-id table DDIT is searched to determine the corresponding descriptor or descriptors for each predicate of a query in the case of a non-insert request and for each keyword of a record in the case of an insert request. In the sequel, we shall refer to this phase as the descriptor search phase and we shall refer to the processing performed therein as descriptor processing. In the next step, the augmented CDT is searched and the corresponding single cluster in the case of an insert request or the corresponding set of clusters in the case of a non-insert request is determined. Once the cluster or cluster set is determined, the authorized cluster or clusters may be selected on the basis of security-related information in the augmented CDT. By searching the same augmented CDT, the addresses of authorized cluster(s) can be found. We shall refer to this step, in the sequel, as the address generation phase and we shall refer to the processing performed therein as address generation. Thus, the three phases of directory management are now consolidated in two.

Since the descriptor search phase and the address generation phase are similar for both insert and non-insert requests, in the sequel we shall only consider these two phases for non-insert requests. The discussion extends in a straightforward manner to insert requests.

4.2.2 Processing Strategies for Multiple Back-ends

In previous discussions, we make no distinction whether the two phases were to be carried out in a single computer or in multiple computers (a controller and several back-ends). It is now necessary to discuss how these two

phases will be executed in the controller and multiple back-ends of MDBS. We have identified for MDBS six different strategies for carrying out the descriptor search phase in the multiple back-ends and one strategy for carrying out the descriptor search phase in the controller, thereby a total of seven strategies are identified for various ways of carrying out the descriptor search phase. We have also identified two strategies, one for carrying out the address generation phase in the controller, the other in the back-ends. For completeness, we also consider an eighth strategy for directory management in which both phases are carried out in the controller. These eight strategies for directory management are proposed and evaluated in the next sections.

Let us first, however, indicate our preference for a strategy in which the address generation phase is carried out in multiple back-ends rather than in the controller. By carrying out this phase in the back-ends, MDBS will be alleviated from the controller limitation problem. Since this phase deals with the generation of secondary memory addresses, each back-end would need to generate only those secondary memory addresses associated with that back-end. On the other hand, if the addresses were to be generated by the controller, the controller would need to generate all the relevant secondary memory addresses associated with the entire back-ends. Thus, it is easy to see that the work of address generation is evenly divided up among the back-ends in the former case. This is essential if we are to achieve an ideal system in which the response time is inversely proportional to the number of back-ends. This concludes our discussion of our preference for a strategy in which the address generation phase is carried out in the multiple back-ends.

We note that the address generation phase actually includes all security related processing also. The time for address generation, which is essentially the time for searching the augmented CDT, will depend on the size of each entry in the augmented CDT. Hence, it will depend on whether or not the table is augmented with security information. In the sequel, our analysis will assume that no security information is contained in the CDT. There are two reasons for this assumption.

First, we wish to analyze the performance of an MDBS in which security is not enforced. This is because many implementations of MDBS may wish to provide only the basic database management functions and may not wish to provide security enforcement.

Second, the enforcement of security will not affect our comparative study

of the various strategies, since all the back-ends are expected to spend the same amount of work in security-related processing.

We now proceed to discuss the eight different strategies for directory management in the following sections. We propose various strategies for directory management in terms of descriptor searching and address generation. These alternatives will be evaluated from the viewpoint of performance and storage requirements.

A. The Centralized Strategy

In this strategy, all the directory management is done at the controller. The controller maintains the descriptors for all the directory attributes. In other words, both DDIT and the augmented CDT are stored with the controller. Given a query, the controller first performs the descriptor processing and then address generation by utilizing the aforementioned tables. Eventually, a set of secondary memory addresses of relevant records is generated at the controller. We note that a secondary memory address consists of not just the track and cylinder information about the records but also the information about the back-end in which the track and cylinder are located. None of the remaining seven strategies will need the back-end information in a secondary memory address. This is because all the remaining strategies will do the address generation at the respective back-end rather than at the controller.

B. The Partially Centralized Strategy

The descriptor processing is done at the controller in this strategy as in the previous strategy. The corresponding descriptors are then broadcasted to all the back-ends. Each back-end will now carry out the address generation phase in an independent fashion. The reason why we expect this strategy to be superior to the previous strategy is two-fold. First, the work of address generation that could be done at the controller is now distributed to the back-ends. This should alleviate the controller limitation problem. Second, the work needed for address generation is divided in such a way that a back-end needs only to generate the addresses of the secondary memory of that back-end.

In this strategy, the descriptor search phase of the directory management is still performed at the controller. The six remaining strategies we will consider are those which try to diminish the effect of having this descriptor search phase performed solely at the controller. In other words, the following stra-

tegies will alleviate the controller limitation problem further.

C. The Rotating Strategy

In the rotating strategy, the descriptor processing during the descriptor search phase is done in a round-robin fashion among the controller and the back-ends. More specifically, the first query is processed at the controller, the second query is processed at the first back-end, the third query is processed at the second back-end, and so on. As a result, it is hoped that some alleviation of the controller limitation problem will take place. As in the partially centralized strategy, the address generation is done individually at each back-end. When the arrival rate of requests to MDBS is low, this strategy will not perform any better than the partially centralized strategy. On the other hand, when the arrival rate of request is high, this strategy may lead to an improvement in performance over the partially centralized strategy. This is because the descriptor processing on a number of queries by multiple back-ends may be overlapped while the descriptor processing on individual queries by the controller must be done sequentially in the partially centralized strategy.

D. The Rotating Without Controller Strategy

This strategy is very similar to the previous one. The only difference is that no descriptor processing is done at the controller. Thus, the descriptor processing for the first query is done at the first back-end, the descriptor processing for the second query is done at the second back-end, and so on. After descriptor processing is completed at a back-end, the corresponding descriptors must be broadcast to all back-ends so that they may proceed with the address generation phase. The only reason for introducing this strategy into consideration is that it would appear to alleviate the controller limitation problem completely from directory management.

We note that both the rotating with controller and the rotating without controller strategies require the duplication of the necessary tables (i.e., DDIT) at all back-ends. This is tolerated for the following reasons. It will be shown, later, that the tables needed for address generation are very much larger than the tables needed for descriptor processing. As a result, duplicating the tables needed for descriptor processing for multiple back-end is tolerable. Furthermore, we are willing to sacrifice storage, if it means an improvement in performance.

Up to this point, two of the strategies allow queries to be processed parallelly in the descriptor search phase. There is no parallel descriptor processing of predicates for a given query. In the following three strategies, we explore the possibility of parallel descriptor processing of predicates for individual queries during the descriptor search phase.

E. The Fully Duplicated Strategy

In the fully duplicated strategy, the descriptor processing is done across the back-ends. More specifically, if there are n back-ends in MDBS and a query contains x predicates, each back-end will process x/n predicates and generate x/n corresponding descriptors which will, in turn, be communicated to each other. Each back-end may then proceed to the address generation phase. Such a strategy also requires the necessary tables to be duplicated at each back-end.

F. The Descriptors Dividing by Attribute Strategy

In this strategy, we explore the possibility of achieving parallel descriptor processing without the need for any duplication of necessary tables. If there are i directory attributes and n back-ends in MDBS, each back-end will maintain the descriptors corresponding to i/n attributes. Each back-end will process those predicates in a query where the descriptors corresponding to the attribute in the predicate is maintained at that back-end. It is expected that each back-end will do an equal amount of descriptor processing, although there may be cases where one back-end does more descripter processing than the others. This happens if all the predicates in a query are such that the descriptors that need to be searched are all stored at the same back-end.

G. The Descriptors Division Within Attribute Strategy

Like the previous strategy, this strategy also attempts parallel descriptor processing without any duplication of the necessary tables. If there are i descriptors on each directory attribute, each back-end will maintain for each attribute i/n descriptors. Thus, descriptor processing is spread over the back-ends. All back-ends will participate in the descriptor processing of a query. After ea back-end obtains some results, they exchange their results. Then, each back-end preceeds with its own address generation phase.

H. The Fully Replicated Strategy

This is the final strategy we consider for doing directory management. In

this strategy, each back-end will work on the entire query during the descriptor search phase. The advantage of letting each back-end do the descriptor processing on all queries is that, unlike the previous three strategies, exchanges among back-ends are unnecessary in this strategy because all back-ends have all the needed results. After completing the descriptor processing, each back-end does its address generation.

4.2.3 Performance Evaluation of the Directory Management Strategies

In this section, we compare the eight strategies for directory management on the basis of performance. For our convenience, we name these strategies A through H, respectively.

Two different approaches are used for the performance analysis. The first approach considers the directory management process alone. It does not consider the fact that there may be queues at the various back-ends and that the controller may allow overlapping of query handling for directory management. Specifically, the different strategies are compared in terms of the time duration from the receipt of a request to the point where all the necessary secondary memory addresses are generated, including the time taken for messages exchanges.

The second approach studies the relative response time for a typical request when each of these eight strategies is employed for directory management. In other words, the directory management is considered in terms of its effect on the overall response time of a request. This study employs a closed queueing network model of MDBS.

One issue that is important to both approaches is whether the necessary tables for descriptor processing and address generation can be stored in the main memory. In our analysis, we assume that a page of the descriptor-to-descriptor-id table DDIT is in the main memory with probability p, where p is high. This is because DDIT is small. For the augmented cluster definition table, i.e., the augmented CDT, on the other hand, we assume that only a certain amount of the main memory, m, is reserved for the table. Now, if the augmented CDT requires greater amount of memory, g, then pages of the augmented CDT are assumed to be in the main memory with probability $\frac{m}{g}$. In general, $\frac{m}{g} < p$.

Another issue to be resolved before we begin our analyses concerns the searching of the descriptors. It is clearly possible to store the descriptors in sorted order and search for the right descriptor using a binary search. Another technique would be to store the descriptors as a B-tree, whose leaf nodes

are the descriptors themselves. However, since the number of descriptors is not expected to be very large, we shall assume that a binary search is used.

A. Time Analyses and Performance Equations

In this section, we present the results for the execution time of the directory management algorithms from the point that a query is received to the point where the addresses of the records are generated and made available at the back-ends. This time therefore includes the time for descriptor processing and address generation.

The following observations may be used to simplify our calculations. First, the centralized strategy and the partially centralized strategy differ only in the address generation phases. Clearly, the partially centralized strategy takes less time for address generation and, hence, is superior to the centralized strategy. We only need to compare the remaining strategies to determine the best one. All the remaining strategies use exactly the same algorithm for address generation. Hence, the time for address generation need not be included in our comparison study. With these observations, let us proceed with our comparison study. Let,

tm: time to send a message from (to) the controller
D: total number of descriptors
i: total number of directory attributes
td: time to read a descriptor from the main memory
tp: time to read an entry from the DDIT in the main memory.
ta: time to read a predicate
x: number of predicates in a query conjunction
tpr: time for an arithmetic operation
k: number of descriptors per secondary memory page
tpg: time to access a secondary memory page
lg: logarithm of the base 2
u(z): the nearest integer greater than or equal to z
tb: time taken for doing a binary search on k descriptors

In the ensuing discussions, let us calculate the total time taken in order to complete the descriptor search phase and have the corresponding set of descriptors available at all back-ends. That is, let us calculate the time taken for directory processing and the time for exchanging any messages that may be needed. For simplicity, we assume that users only employ single conjunctions

in their requests. If a disjunction of p conjunctions is used in a request, it may be easily treated as p separate requests.

In Strategies A, B, C, D and H, it is easy to see that the descriptor processing on the entire query conjunction is done at a single computer. Thus, the average-case time is expressed as follows

$$x(ta + (i/2)(tp + tpr) + lg((td + 4tpr)D/i) + 2tpr) \text{ ----- } (1)$$

Here is the explanation. For each of the x predicates in the conjunction, the following must be done. First, the predicate must be read and this takes ta time units. Next, the i entries in DDIT must be searched. On the average, i/2 entries will need to be searched. Each of the i/2 entries must be read (taking tp time units each time) and compared with the attribute in the predicate (taking tpr time units). Next, the set of D/i descriptors must be searched. Our algorithm for a binary search of N descriptors is included in appendix D. From the algorithm, the total time to do binary search on N descriptor is

$$2tpr + lgN(\text{time to read a descriptor} + 4tpr)$$

Thus, the time taken for doing a binary search on D/i descriptors will take

$$2tpr + lg((td + 4tpr)D/i)$$

Together, we obtain equation (1).

In arriving at equation (1), we assumed that all the descriptors are in the main memory. Let us now improve equation (1) by assuming that only a fraction p of the descriptor pages are in the main memory. We assume that all the D descriptors are stored in secondary-memory pages each of which contains up to k descriptors. Also, descriptors for different attributes are stored in different pages. In searching descriptors organized as pages, we assume the following algorithm is used. Pages are retrieved sequentially. For each page retrieved, the ranges of the first and last descriptors in the page are compared to the value in the predicate. This will tell us if the page contains the descriptor we are looking for. If so, a binary search of that page is performed. If not, the next page is retrieved, and so on. Since the first page must be processed before the second one is brought in and because the pages needed may not be adjacent in the secondary memory, we assume no I/O overlap with CPU processing. Then, the time for descriptor processing is:

$$x(ta + (i/2)(tp + tpr) + (u(D/(2ik)) - 1)((1 - p)tpg + 2(td + tpr)) + ((1 - p)tpg + 2(td + tpr) + tb)) \text{ ---------- } (2)$$

in the average case. The worst-case time may be obtained from equation (2) by replacing u(D/(2ik)) with u(D/(ik)) and (i/2)(tp + tpr) with i(tp + tpr). Here, the time taken for doing a binary search on k descriptors, tb, is (2tpr + lg k (td + 4tpr)).

Let tdr be the time taken for descriptor processing. Then tdr is equal to above equation (2). Finally, the time taken for descriptor processing and message exchanging in strategies A, B and H is

$$(tm + tdr)$$

where tm is the time taken to broadcast the corresponding descriptors from the controller to the back-ends in strategies A and B, tm is the time to broadcast the original query conjunction to all the back-ends each of which individually spends tdr time units to calculate the corresponding descriptors in Strategy H.

In Strategy C, the time for descriptor processing and message exchanging is

$$(n/(n + 1)) * (tdr + 2tm) + (1/(n + 1)) * (tdr + tm)$$

This is explained as follows for a system with n back-ends. If the processing is done in the controller and happens once every (n+1) times, the processing must include the time for braodcasting the corresponding descriptors from the controller to the back-ends. On the other hand, if the processing is done at one of the back-ends as happens n out of (n + 1) times, then the processing time must include the time for sending the original query conjunction to a back-end and the time for broadcasting the corresponding descriptors to all back-ends from that back-end.

Finally, in Strategy D, the time for descriptor processing and message exchanging is

$$(tdr + 2tm)$$

We now need to calculate the descriptor processing time for strategies E, F and G. For Strategy E, the time for descriptor processing in the average case is:

$$u(x/n)(ta + (i/2)(tp + tpr) + ((1 - p)tpg + 2(td + tpr) + tb) + (u(D/(2ik)) - 1)((1 - p)tpg + 2(td + tpr))) + 2tm \quad \text{---------- (3)}$$

The time for descriptor processing in the worst case may be obtained from equation 3 by replacing (i/2)(td + tpr) with i(tp + tpr) and u(D/(2ik)) with u(D/(ik)). Time for two-message exchanges is also included in the above equation

One of the messages is for braodcasting the original query conjunction from the controller to all the back-ends. The second one is for exchanging partial results among the back-ends. We note that this method requires the presence of all the descriptors at each back-end. Furthermore, no more than x back-ends can be executing a query conjunction simultaneously, since there are x predicates in the query conjunction. This is the maximum degree of parallelism that can be brought to bear on the processing of descriptors.

Let us now present the descriptor processing and message exchange time for Strategy F as follows.

$$u(x/n)(ta + u(i/(2n))(tp + tpr) + ((1-p)tpg + 2(td + tpr) + tb) + (u(D/(2ik)) - 1)((1-p)tpg + 2(td + tpr))) + 2tm \quad \text{---------- (4)}$$

The worst case time is obtained from equation by replacing u(x/n) with x, u(i/(2n)) with u(i/n), and D/(2ik) with D/(ik). In general, however, we expect this strategy to perform at the time closer to the average-case time than the worst-case time. This is because the database administrator may use his knowledge of the typical query conjunctions to assign descriptors to back-ends in an appropriate fashion.

Finally, the average-case time for descriptor processing and message exchange by Strategy G is as follows.

$$x(ta + u(i/2)(tp + tpr) + ((1-p)tpg + 2(td + tpr) + tb) + (u(D/(2nik)) - 1)((1-p)tpg + 2(td + tpr))) + 2tm \quad \text{---------- (5)}$$

The worst-case time is obtained from equation 5 by replacing u(i/2) with i and D/(2nik) with D/(nik). It is clear that for small values of D and large values of k, increasing n is going to have little effect in reducing the time. For instance, consider that the number of descriptors, D, is 100 and that the number of descriptors stored in a page, k, is 100. Then u(D/(nik)) = 1, irrespective of the number of back-ends, n. Since u(D/(nik)) is the only expression in equation (5) which contains n, it is clear that increasing n is not going to reduce the time for descriptor processing and message exchange. However, this strategy should prove advantageous for large values of D.

B. Computations and Their Interpretations Resulted from the Performance Equations

In order to compare the descriptor processing and message exchange times of MDBS under various strategies, we use the following values for the parameters

in the equations (1) through (5) to calculate the times.

tm	= 8msecs	(time to send a message from (to) the controller)
tpr	= 5μsecs	(time for an arithmetic operation)
td	= 5μsecs	(time to read a descriptor from the main memory)
ta	= 3μsecs	(time to read a predicate)
tp	= 3μsecs	(time to read an entry from the DDIT in the main memory)
tpg	= 47.3msecs	(time to access a secondary memory page)
k	= 85	(no. of descriptors per secondary memory page)
p	= 0.9	(probability of a page of DDIT entries being in the main memor
x	= 5	(no. of predicates in a query conjunction)

The value of n, the number of back-ends, is chosen from the set {2, 5, 8}. The ratio D/i, the number of descriptors per directory attribute, is chosen from the set {10, 20, 30, 40}. The number of directory attributes is chosen from the set {5, 10, 15}.

The results of our calculations are shown in Figure 16. In this figure, we present two rows of results for each number of back-ends. The first row gives the average-case times and the second row gives the worst-case times for descriptor processing and message exchanging. It is seen that two of the three strategies that utilize parallel processing of predicates of a given query conjunction during the descriptors search phase, namely strategies E and F, give the best average-case results. They consistently outperform all the other strategies over the entire range of variation which we tried for the number of back-ends, the number of directory attributes and the number of descriptors per attribute. The times under strategies E and F may be as much as 12 msecs less than the time under the next best strategy. This occurs when the number of back-ends employed is eight.

Looking at the worst-case results, we see that Strategy E is, once again, the best strategy. The worst-case performance of Strategy F, however, is far worse than that of Strategy E. This is because the worst case for Strategy F occurs when all the predicates in a query conjunction are such that the descriptors that need to be searched for descriptor processing are all stored at a single back-end. Thus, all the descriptor processing is performed at this single back-end and parallel descriptor precessing is not achieved. In Strategy E, on the other hand, the duplication of descriptors allows us to achieve parallel descriptor processing for all types of query conjunctions.

In addition, the following observations may be made from the results of Figure 16. Under Strategies E and F, the performance of MDBS improves with an increase in the number of back-ends. However, this improvement will not go further if the number of back-ends is greater than the number of predicates in a query

Number of Attributes = 5
Number of Descriptors Per Attribute = 10

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.74	32.74	38.07	40.74	30.84	30.82	40.74	32.74
	32.82	32.82	38.15	40.82	30.89	40.74	40.82	32.82
5	32.74	32.74	39.40	40.74	20.95	20.93	40.74	32.74
	32.82	32.82	39.48	40.82	20.96	40.66	40.82	32.82
8	32.74	32.74	39.85	40.74	20.95	20.93	40.74	32.74
	32.82	32.82	39.93	40.82	20.96	40.66	40.82	32.82

Number of Attributes = 5
Number of Descriptors Per Attribute = 20

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.74	32.74	38.07	40.74	30.84	30.82	40.74	32.74
	32.82	32.82	38.15	40.82	30.89	40.74	40.82	32.82
5	32.74	32.74	39.40	40.74	20.95	20.93	40.74	32.74
	32.82	32.82	39.48	40.82	20.96	40.66	40.82	32.82
8	32.74	32.74	39.85	40.74	20.95	20.93	40.74	32.74
	32.82	32.82	39.93	40.82	20.96	40.66	40.82	32.82

Number of Attributes = 5
Number of Descriptors Per Attribute = 30

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.74	32.74	38.07	40.74	30.84	30.82	40.74	32.74
	32.82	32.82	38.15	40.82	30.89	40.74	40.82	32.82
5	32.74	32.74	39.40	40.74	20.95	20.93	40.74	32.74
	32.82	32.82	39.48	40.82	20.96	40.66	40.82	32.82
8	32.74	32.74	39.85	40.74	20.95	20.93	40.74	32.74
	32.82	32.82	39.93	40.82	20.96	40.66	40.82	32.82

Note: There are two rows corresponding to each value of the number of back-ends. The first row gives the average-case times and the second row gives the worst-case times.

Figure 16. Directory-Processing-and-Message-Exchanging Times (in msecs) Under Different Strategies

Number of Attributes = 5
Number of Descriptors Per Attribute = 40

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.74	32.74	38.07	40.74	30.84	30.82	40.74	32.74
	32.82	32.82	38.15	40.82	30.89	40.74	40.82	32.82
5	32.74	32.74	39.40	40.74	20.95	20.93	40.74	32.74
	32.82	32.82	39.48	40.82	20.96	40.66	40.82	32.82
8	32.74	32.74	39.85	40.74	20.95	20.93	40.74	32.74
	32.82	32.82	39.93	40.82	20.96	40.66	40.82	32.82

Number of Attributes = 10
Number of Descriptors Per Attribute = 10

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.82	32.82	38.15	40.82	30.89	30.84	40.82	32.82
	33.02	33.02	38.35	41.02	31.01	40.82	41.02	33.02
5	32.82	32.82	39.48	40.82	20.96	20.93	40.82	32.82
	33.02	33.02	39.68	41.02	21.00	40.70	41.02	33.02
8	32.82	32.82	39.93	40.82	20.96	20.93	40.82	32.82
	33.02	33.02	40.13	4.102	21.00	40.70	41.02	33.02

Number of Attributes = 10
Number of Descriptors Per Attribute = 20

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.82	32.82	38.15	40.82	30.89	30.84	40.82	32.82
	33.02	33.02	38.35	41.02	31.01	40.82	41-02	33.02
5	32.82	32.82	39.48	40.82	20.96	20.93	40.82	32.82
	33.02	33.02	39.68	41.02	21.00	40.70	41.02	33.02
8	32.82	32.82	39.93	40.82	20.96	20.93	40.82	32.82
	33.02	33.02	40.13	41.02	21.00	40.70	41.02	33.02

Number of Attributes = 10
Number of Descriptors Per Attribute = 30

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.82	32.82	38.15	40.82	30.89	30.84	40.82	32.82
	33.02	33.02	38.35	41.02	31.01	40.82	41.02	33.02
5	32.82	32.82	39.48	40.82	20.96	20.93	40.82	32.82
	33.02	33.02	39.68	41.02	21.00	40.70	41.02	33.02
8	32.82	32.82	39.93	40.82	20.96	20.93	40.82	32.82
	33.02	33.02	40.13	41.02	21.00	40.70	41.02	33.02

Figure 16. (Contd.)

Number of Attributes = 10
Number of Descriptors Per Attribute = 40

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.82	32.82	38.15	40.82	30.89	30.84	40.82	32.82
	33.02	33.02	38.35	41.02	31.01	40.82	41.02	33.02
5	32.82	32.82	39.48	40.82	20.96	20.93	40.82	32.82
	33.02	33.02	39.68	41.02	21.00	40.70	41.02	33.02
8	32.82	32.82	39.93	40.82	20.96	20.93	40.82	32.82
	33.02	33.02	40.13	41.02	21.00	40.70	41.02	33.02

Number of Attributes = 15
Number of Descriptors Per Attribute = 10

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.94	32.94	38.27	40.94	30.96	30.87	40.94	32.94
	33.22	33.22	38.55	41.22	31.13	40.94	41.22	33.22
5	32.94	32.94	39.60	40.94	20.99	20.94	40.94	32.94
	33.22	33.22	39.88	41.22	21.04	40.74	41.22	33.22
8	32.94	32.94	40.05	40.94	20.99	20.93	40.94	32.94
	33.22	33.22	40.33	41.22	21.04	40.70	41.22	33.22

Number of Attributes = 15
Number of Descriptors Per Attribute = 20

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.94	32.94	38.27	40.94	30.96	30.87	40.94	32.94
	33.22	33.22	38.55	41.22	31.13	40.94	41.22	33.22
5	32.94	32.94	39.60	40.94	20.99	20.94	40.94	32.94
	33.22	33.22	39.88	41.22	21.04	40.74	41.22	33.22
8	32.94	32.94	40.05	40.94	20.99	20.93	40.94	32.94
	33.22	33.22	40.33	41.22	21.04	40.70	41.22	33.22

Number of Attributes = 15
Number of Descriptors Per Attribute = 30

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.94	32.94	38.27	40.94	30.96	30.87	40.94	32.94
	33.22	33.22	38.55	41.22	31.13	40.94	41.22	33.22
5	32.94	32.94	39.60	40.94	20.99	20.94	40.94	32.94
	33.22	33.22	39.88	41.22	21.04	40.74	41.22	33.22
8	32.94	32.94	40.05	40.94	20.99	20.93	40.94	32.94
	33.22	33.22	40.33	41.22	21.04	40.70	41.22	33.22

Figure 16. (Contd.)

Number of Attributes = 15
Number of Descriptors Per Attribute = 40

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.94	32.94	38.27	40.94	30.96	30.87	40.94	32.94
	33.22	33.22	38.55	41.22	31.13	40.94	41.22	33.22
5	32.94	32.94	39.60	40.94	20.99	20.94	40.94	32.94
	33.22	33.22	39.88	41.22	21.04	40.74	41.22	33.22
8	32.94	32.94	40.05	40.94	20.99	20.93	40.94	32.94
	33.22	33.22	40.33	41.22	21.04	40.70	41.22	33.22

Figure 16. (Contd.)

conjunction. The performance of MDBS under other strategies, do not improve with increasing number of back-ends. Thus, these other strategies are not suitable for implementation in MDBS.

The results of Figure 16 do not clearly indicate whether performance of MDBS under any strategy is independent of the number of descriptors per directory attribute. To test whether the performance of MDBS is independent of the number of descriptors per attribute, we make a second set of calculations with the number of descriptors per attribute varying between 100 and 200. The results are shown in Figure 17. Comparing corresponding entries in Figure 16 with the ones in Figure 17, we see that the increasing number of descriptors per attribute affects the performance considerably. Also, we see that Strategies E and F provide the best average-case results. In fact, differences as large as 30 msecs are now possible between these two strategies and the remaining strategies. Interestingly enough, the performance of Strategy G is now comparable to that of Strategies E and F.

We make a final set of calculations with the number of descriptors per attribute varying from 700 to 900. Of course, this may be an unreasonably large range of values for the number of descriptors per attribute. However, we are interested in the results of this calculation from a point of view where MDBS is heavily loaded and utilized. The results, shown in Figure 18, indicate that Strategy G is now comparable to, and occasionally even better than, Strategies E and F. By and large, these three strategies which employ parallel processing of predicates by multiple back-ends during the descriptor search phase outperform the other strategies.

C. A Preliminary Conclusion Based on the Performance Equations

The results of our study may be summarized as follows. The three Strategies - namely, Strategies E, F and G, which utilize parallel processing of predicates of a query conjunction during the descriptor search phase, may provide better performance than the other five strategies. Furthermore, the employment of any of these strategies in MDBS leads to an improvement in performance with each increase in the number of back-ends. However, the extremely poor worst-case performance of Strategy F would eliminate it from consideration for MDBS implementation. Similarly, the poor average-and-worst case performance of Strategy G for typical values of number of attribute and number of descriptors per attribute would eliminate it from consideration for MDBS implementation. Consequently,

Number of Attributes = 5
Number of Descriptors Per Attribute = 100

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.74	32.74	38.07	40.74	30.84	30.82	40.74	32.74
	56.57	56.57	61.90	64.57	45.14	64.49	40.82	56.57
5	32.74	32.74	39.40	40.74	20.95	20.93	40.74	32.74
	56.57	56.57	63.23	64.57	25.71	64.41	40.82	56.57
8	32.74	32.74	39.85	40.74	20.95	20.93	40.74	32.74
	56.57	56.57	63.68	64.57	25.71	64.41	40.82	56.57

Number of Attributes = 5
Number of Descriptors Per Attribute = 200

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	56.49	56.49	61.82	64.49	45.09	45.07	40.74	56.49
	80.32	80.32	85.65	88.32	59.39	88.24	40.82	80.32
5	56.49	56.49	63.15	64.49	25.70	25.68	40.74	56.49
	80.32	80.32	86.98	88.32	30.46	88.16	40.82	80.32
8	56.49	56.49	63.60	64.49	25.70	25.68	40.74	56.49
	80.32	80.32	87.43	88.32	30.46	88.16	40.82	80.32

Number of Attributes = 5
Number of Descriptors Per Attribute = 300

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	56.49	56.49	61.82	64.49	45.09	45.07	40.74	56.49
	104.07	104.07	109.40	112.07	73.64	111.99	40.82	104.07
5	56.49	56.49	63.15	64.49	25.70	25.68	40.74	56.49
	104.07	104.07	110.73	112.07	35.21	111.91	40.82	104.07
8	56.49	56.49	63.60	64.49	25.70	25.68	40.74	56.49
	104.07	104.07	111.18	112.07	35.21	111.91	40.82	104.07

Note: There are two rows corresponding to each value of the number of back-ends. The first row gives the average case times and the second row gives the worst case times.

Figure 17. Descriptor Processing and Message Exchanging Times (in msecs) for Various Directory Management Strategies

Number of Attributes = 10
Number of Descriptors Per Attribute = 100

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.82	32.82	38.15	40.82	30.89	30.84	40.82	32.82
	56.77	56.77	62.10	64.77	45.26	64.57	41.02	56.77
5	32.82	32.82	39.48	40.82	20.96	20.93	40.82	32.82
	56.77	56.77	63.43	64.77	25.75	64.45	41.02	56.77
8	32.82	32.82	39.93	40.82	20.96	20.93	40.82	32.82
	56.77	56.77	63.88	64.77	25.75	64.45	41.02	56.77

Number of Attributes = 10
Number of Descriptors Per Attribute = 200

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	56.57	56.57	61.90	64.57	45.14	45.09	40.82	56.57
	80.52	80.52	85.85	88.52	59.51	88.32	41.02	80.52
5	56.57	56.57	63.23	64.57	25.71	25.68	40.82	56.57
	80.52	80.52	87.18	88.52	30.50	88.20	41.02	80.52
8	56.57	56.57	64.68	64.57	25.71	25.68	50.82	56.57
	80.52	80.52	87.63	88.52	30.50	88.20	41.02	80.52

Number of Attributes = 10
Number of Descriptors Per Attribute = 300

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	56.57	56.57	61.90	64.57	45.14	45.09	40.82	56.57
	104.27	104.27	109.60	112.27	73.76	112.07	41.02	104.27
5	56.57	56.57	63.23	64.57	25.71	25.68	40.82	56.57
	102.37	104.27	110.93	112.27	35.25	111.95	41.02	104.27
8	56.57	56.57	63.68	64.57	25.71	25.68	40.82	56.57
	104.27	104.27	111.38	112.27	35.25	111.95	41.02	104.27

Number of Attributes = 15
Number of Descriptors Per Attribute = 100

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	32.94	32.94	38.27	40.94	30.96	30.87	40.94	32.94
	56.97	56.97	62.30	64.97	45.38	64.69	41.22	56.97
5	32.94	32.94	39.60	40.94	20.99	20.94	40.94	32.94
	56.97	56.97	63.63	64.97	25.79	64.49	41.22	56.97
8	32.94	32.94	40.05	40.94	20.99	20.93	40.94	32.94
	56.97	56.97	64.08	64.97	25.79	64.45	41.22	56.97

Figure 17. (Contd.)

Number of Attributes = 15
Number of Descriptors Per Attribute = 200

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	56.69	56.69	62.02	64.69	45.21	45.21	40.94	56.69
	80.72	80.72	86.05	88.72	59.63	88.44	41.22	80.72
5	56.69	56.69	63.35	64.69	25.74	25.69	40.94	56.69
	80.72	80.72	87.38	88.72	30.54	88.24	41.22	80.72
8	56.69	56.69	63.80	64.69	25.74	25.68	40.94	56.69
	80.72	80.72	87.83	88.72	30.54	88.20	41.22	80.72

Number of Attributes = 15
Number of Descriptors Per Attribute = 300

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	56.69	56.69	62.02	64.69	45.21	45.12	40.94	56.69
	104.47	104.47	109.80	112.47	73.88	112.19	41.22	104.47
5	56.59	56.69	63.35	64.69	25.74	25.69	40.94	56.69
	104.47	104.47	111.13	112.47	35.29	111.99	41.22	104.47
8	56.69	56.69	63.80	64.69	25.74	25.68	40.94	56.69
	104.47	104.47	111.58	112.47	35.29	111.95	41.22	104.47

Figure 17. (Contd.)

Number of Attributes = 5
Number of Descriptors Per Attribute = 700

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	127.74	127.74	133.07	135.74	87.84	87.82	88.24	127.74
	222.82	222.82	228.15	230.82	144.89	230.74	88.32	222.82
5	127.74	127.74	134.40	135.74	39.95	39.93	40.74	127.74
	222.82	222.82	229.48	230.82	58.96	230.66	40.82	222.82
8	127.74	127.74	134.85	135.74	39.95	39.93	40.74	127.74
	222.82	222.82	229.93	230.82	58.96	230.66	40.82	222.82

Number of Attributes = 5
Number of Descriptors Per Attribute = 800

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	127.74	127.75	133.07	135.74	87.84	87.82	88.24	127.74
	246.57	246.57	251.90	254.57	159.14	254.49	88.32	246.57
5	127.74	127.74	134.40	135.74	39.95	39.93	40.74	127.74
	246.57	246.57	253.23	254.57	63.71	254.41	40.82	246.57
8	127.74	127.74	134.85	135.74	39.95	39.93	40.74	127.74
	246.57	246.57	253.68	254.57	63.71	254.41	40.82	246.57

Number of Attributes = 5
Number of Descriptors Per Attribute = 900

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	151.49	151.49	156.82	159.49	102.09	102.07	88.24	151.49
	270.32	270.32	275.65	278.32	173.39	278.24	88.32	270.32
5	151.49	151.49	158.15	159.49	44.70	44.68	64.49	151.49
	270.32	270.32	276.98	278.32	68.46	278.16	64.57	270.32
8	151.49	151.49	158.60	159.49	44.70	44.68	40.74	151.49
	270.32	270.32	277.43	278.32	68.46	278.16	40.82	270.32

Note: There are two rows for each value of the number of back-ends. The first row gives the average case times and the second row gives the worst case times.

Figure 18. Descriptor-Processing-and-Message-Exchange Times (in msecs) for Various Directory Management Strategies

Number of Attributes = 10
Number of Descriptors Per Attribute = 700

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	127.82	127.82	133.15	135.82	87.89	87.84	88.32	127.82
	223.02	223.02	228.35	231.02	145.01	230.82	88.52	223.02
5	127.82	127.82	134.48	135.82	39.96	39.93	40.82	127.82
	223.02	223.02	229.68	231.02	59.00	230.70	41.02	223.02
8	127.82	127.82	134.93	135.82	39.96	39.93	40.82	127.82
	223.02	223.02	230.13	231.02	59.00	230.70	41.02	223.02

Number of Attributes = 10
Number of Descriptors Per Attribute = 800

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	127.82	127.82	133.15	135.82	87.89	87.84	88.32	127.82
	246.77	246.77	252.10	254.77	159.26	254.57	88.52	246.77
5	127.82	127.82	134.48	135.82	39.96	39.93	41.82	127.82
	246.77	246.77	253.43	254.77	63.75	254.45	41.02	246.77
8	127.82	127.82	134.93	135.82	39.96	39.93	40.82	127.82
	246.77	246.77	253.88	254.77	63.75	254.45	41.02	246.77

Number of Attributes = 10
Number of Descriptors Per Attribute = 900

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	151.57	151.57	156.90	159.57	102.14	102.09	88.32	151.57
	270.52	270.52	275.85	278.52	173.51	278.32	88.52	270.52
5	151.57	151.57	158.23	159.57	44.71	44.68	64.57	151.57
	270.52	270.52	277.18	278.52	68.50	278.20	64.77	270.52
8	151.57	151.57	158.68	159.57	44.71	44.68	40.82	151.57
	270.52	270.52	277.63	278.52	68.50	278.20	41.02	270.52

Number of Attributes = 15
Number of Descriptors Per Attribute = 700

Back-ends \ Strategy	A	B	C	D	E	F	G	H
2	127.94	127.94	133.27	135.94	87.96	87.87	88.44	127.94
	223.22	223.22	228.55	231.22	145.13	230.94	88.72	223.22
5	127.94	127.94	134.60	125.94	39.99	39.94	40.94	127.94
	223.22	223.22	229.88	231.22	59.04	230.74	41.22	223.22
8	127.94	127.94	135.05	135.94	39.99	39.93	40.94	127.94
	223.22	223.22	230.33	231.22	59.04	230.70	41.22	223.22

Figure 18. (Contd.)

Number of Attributes = 15
Number of Descriptors Per Attribute = 800

Strategy / Back-ends	A	B	C	D	E	F	G	H
2	127.94	127.94	133.27	135.94	87.96	87.87	88.44	127.94
	267.97	246.97	252.30	254.97	159.38	254.69	88.72	246.97
5	127.94	127.94	134.60	135.94	39.99	39.94	40.94	127.94
	246.97	246.97	253.63	254.97	63.97	254.49	41.22	246.97

Number of Attributes = 15
Number of Descriptors Per Attribute = 900

Strategy / Back-ends	A	B	C	D	E	F	G	H
2	151.69	151.69	157.02	159.69	102.21	102.21	88.44	151.69
	270.72	270.72	276.05	278.72	173.63	278.44	88.72	270.72
5	151.69	151.69	158.35	159.69	44.74	44.69	64.69	151.69
	270.72	270.72	277.38	278.72	68.54	278.24	63.97	270.72
8	151.69	151.69	158.80	159.69	44.74	44.68	40.94	151.69
	270.72	270.72	277.83	278.72	68.54	278.20	41.22	270.72

Figure 18. (Contd.)

the superior strategy for implementation in MDBS is Strategy E.

D. Limitations of Time Analysis and Performance Equations in the Evaluation of Directory Management Strategies

Our previous evaluation of various strategies on the basis of performance equation has a number of limitations. First, the effects of the directory management strategy on other aspects of MDBS are not considered. It is possible that some of the strategies may create bottlenecks at some component of MDBS, thus resulting in very poor overall response times. Second, the effects of queueing delays of requests are neglected. Third, the requirement that Strategies E and F lead to improved performance with an increase in the number of back-ends only when the number of back-ends is no greater than the number of predicates in a query conjunction seems unreasonable. We expect that there will be performance improvement even when the number of back-ends far exceeds the number of predicates in a query conjunction. For example, for Strategy E, F or G, if the number of predicates per query conjunction is five, and there are ten back-ends, two query conjunctions can be handled by MDBS with each back-end processing a predicate. An MDBS with only five back-ends would not be able to handle all the predicates concurrently. Thus, MDBS with five back-ends should perform worse than MDBS with ten back-ends, although the results of our previous study would not indicate this observation. Fourth, the interpretation of the worst-case performance of Strategy F may be misleading. In reality, the performance tends to average out and is close to the performance of the average case. However, looking at the results of Figures 16, 17 and 18, one may be tempted to remove Strategy F from consideration for MDBS implementation. Finally, the strong point of Strategies C and D has not been brought out by the study. Their advantage comes from the fact that multiple queries may be simultaneously executed in the different back-ends. That is, they benefit from inter-query parallelism (where several query conjunctions may be processed in parallel by the back-ends) rather than intra-query parallelism (where only predicates of a query conjunction may be processed in parallel by the back-ends).

E. Performance Analysis Based on a Closed Queueing Network Model

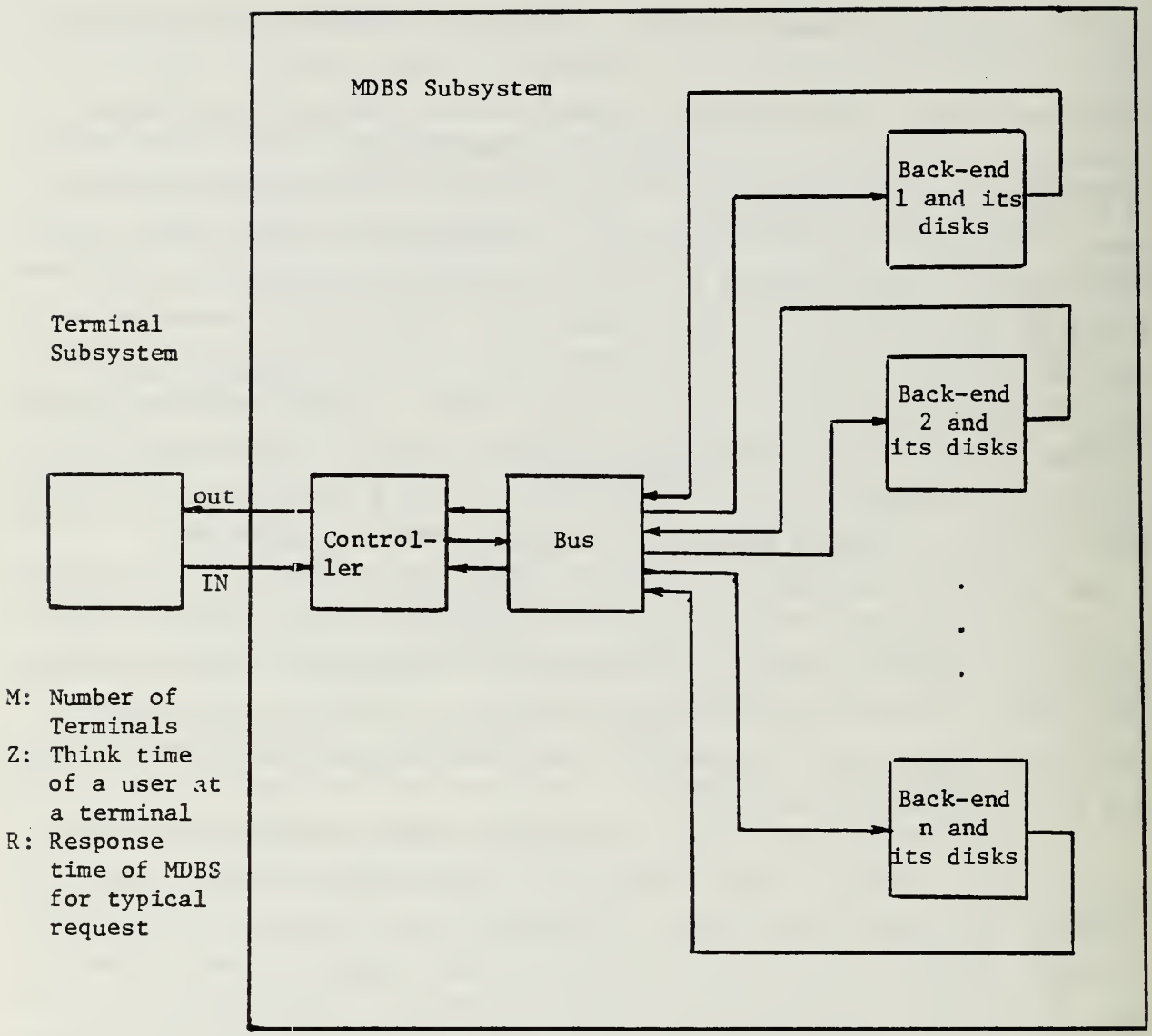
We intend to compare the various directory management strategies by using a closed queueing network model. Such a model overcomes all the limitations of the previous study. First of all, we are going to model all system activities

in MDBS including parsing of requests, descriptor processing, address generation, retrieval of records from the secondary storage, messages passing, and record processing in the back-ends. Even the bus in MDBS will be modelled. Furthermore, such a model takes into account the queueing delays at each point in the system and also accounts for both inter-query and intra-query parallelisms. The model is shown in Figure 19. It consists of two subsystems: the MDBS subsystem (which consists of the controller, the back-ends, the disk drives and the bus) and the terminal subsystem. Each terminal is manned by a user who alternates between thinking and waiting. In the thinking state, the user is contemplating what request to submit next. On submitting a new request to MDBS, the user enters the waiting state, where he will remain until MDBS completes the request execution. The mean time that a user spends in a thinking state is called the think time and is denoted Z. The mean time that a user spends in a waiting state is the response time of MDBS and is denoted by R. Thus, the mean time a user spends at a terminal is (Z+R), since the user is either thinking or waiting at the terminal. Furthermore, R/(Z+R) represents the portion of the time that a user is expected to be in the waiting state. Since there are M users, the number of users who are expected to be in the waiting state is therefore MR/(Z+R).

We note that the time spent in the waiting state is equivalent to the response time required of MDBS. The number of users in the waiting state is therefore equivalent to the number of responses (i.e., requests to be processed in MDBS). From now on, we say the number of user requests in MDBS whenever we mean the number of users who are in the waiting state. Consequently, we consider the MDBS subsystem alone as a closed queueing network model where the number of requests in the subsystem is MR/(Z+R).

Let us now describe the closed queueing network model of MDBS that is used in our study. The various components in a closed queueing network model are usually referred to as devices. Our model of MDBS has 2(n+1) devices. These are the controller, the bus, the n disk systems and the n back-ends.

A separate I/O submodel is used for the disk system. This I/O submodel is an integral part of the overall model and will be discussed in great detail in the following section. The I/O submodel will be used to calculate the response time of the disk system to an I/O request for a track of data.



Arrows show path taken by a typical request during its execution.

Figure 19. Closed Queueing Network Model of MDBS

E.1 The I/O Submodel for Single Requests

We consider the disk system as consisting of M disk drives and a single channel. The I/O submodel consists of M drive queues and one channel queue as shown in Figure 20. It is assumed that requests are independent and arrive randomly in time at the drive queues, each disk drive being likely addressed equally. The inter-arrival distribution is chosen as exponential with L as the mean rate at which requests are received by the disk system. The inter-arrival distribution of requests to each drive queue is also exponential with a mean request rate of L/M. The assumption of exponential distribution is made because it is the most reasonable one in the absence of other information about the inter-arrival distribution.

For a queue, there is a certain rate at which requests will arrive to the queue and a certain rate at which requests in the queue are serviced. It is well-known that a queue may be completely analyzed if we know the mean and variance of the inter-arrival time of requests to the queue and the mean and variance of the service time of requests in the queue. Let us now analyze the drive queues and the channel queue, in turn.

Analyzing a Drive Queue - The drive queue service time is the seek time. The seek time of a disk drive is approximated by an equation of the form $(a + by)$, where a and b are constants and y is the number of cylinders traversed during the seek. We let N be the total number of cylinders in a disk drive. Then,

$$\text{Mean service time} = \text{Mean seek time} = \sum_{y=1}^{N} (a+by)\left(\frac{2}{N} - \frac{2y}{N^2}\right) = a - \frac{a}{N} + \frac{bN}{3} - \frac{b}{3N} .$$

$$\text{Variance of service time} = \text{Variance of seek time} = \frac{b^2N^2}{18} + \frac{b^2}{18} - \frac{a^2}{N^2} - \frac{b^2}{9N^2} - \frac{2ab}{3N^2} + \frac{a^2}{N} + \frac{2ab}{3} .$$

(See Appendix E for the derivation).

For a drive queue, these are the only two quantities we will need in our derivation.

Analyzing the Channel Queue - For the channel queue, the arrival distribution is exponential and the service distribution is constant. Then

$$\text{Mean service time} = \text{disk rotation time} = d .$$

$$\text{Variance of service time} = 0 .$$

$$\text{Mean Inter-arrival time} = \frac{1}{L} \quad \text{(See Figure 20).}$$

$$\text{Variance of Inter-arrival time} = \frac{1}{L^2} .$$

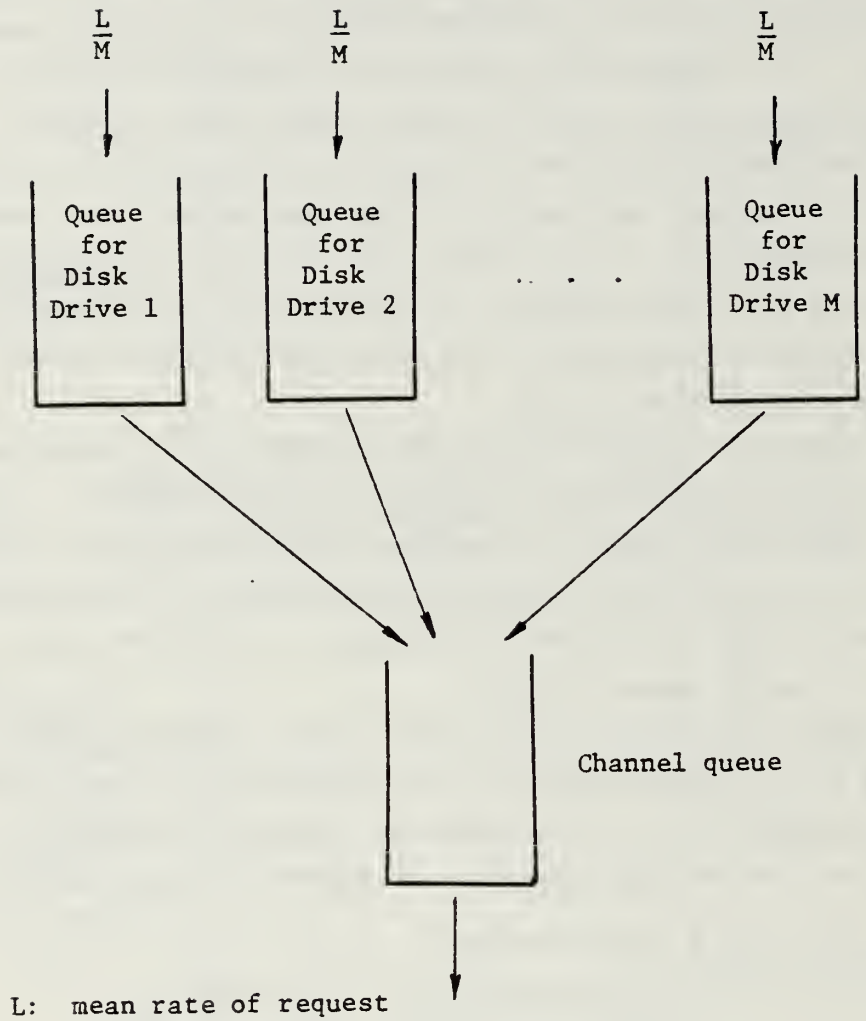


Figure 20. Queueing Model of a Single Channel Disk System

The four quantities needed to analyze the channel queue are, thus, completely specified. Since the inter-arrival distribution for the channel queue is exponential and the service distribution for the channel queue is non-exponential, the channel queue is now analyzed as an M/G/1 queue [Klei75]. Application of a standard queueing theory equation gives us

$$\text{Channel wait time} = \frac{Ld^2}{2(1-Ld)} .$$

Let the time spent by a request in the channel queue and the time spent by a request in getting service after its removal from the channel queue be referred to as the time spent by a request in the channel. Then

$$\text{Mean time in channel} = \frac{Ld^2}{2(1-Ld)} + d .$$

$$\text{Variance of time in channel} = \frac{L^2d^4}{4(1-Ld)^2} + \frac{Ld^3}{3(1-Ld)} .$$

(See Appendix E for the derivation).

This completes our analysis of the channel queue.

Analyzing the Composite Queue - We shall consider a drive queue and channel queue as a composite queue. This is because after a disk drive completes a seek, it must acquire the channel for transfering the track of data over the channel. Only then may a disk drive begin another seek.

Mean service time of Composite Queue

$$= s$$

$$= \text{Mean seek time} + \text{Mean time in channel}$$

$$= a - \frac{a}{N} + \frac{bN}{3} - \frac{b}{3N} + \frac{Ld^2}{2(1-Ld)} + d .$$

Variance of service time of Composite Queue

$$= v$$

$$= \text{Variance of seek time} + \text{Variance of time in channel}$$

$$= \frac{b^2N^2}{18} + \frac{b^2}{18} - \frac{a^2}{N^2} - \frac{b^2}{9N^2} - \frac{2ab}{3N^2} + \frac{a^2}{N} + \frac{2ab}{3} + \frac{L^2d^4}{4(1-Ld)^2} + \frac{Ld^3}{3(1-Ld)} .$$

Mean Inter-arrival time to composite queue

$$= \frac{M}{L} .$$

Variance of Inter-arrival time to composite queue

$$= \frac{M^2}{L^2} .$$

Thus, all the four quantities needed to specify the composite queue are known. Since the inter-arrival distribution of the queue is exponential and the ser-

vice distribution is non-exponential, the composite queue is an M/G/1 queue. Then, using a standard queueing theory equation, the total time spent by a request in the composite queue which is the total time spent by a request in the disk system and, hence, the response time for a request is r and

$$r = \frac{(\frac{L}{M})(s^2 + v)}{2(1 - \frac{Ls}{M})} + s .$$

A Note on this model vs. other models - The I/O submodel that we have developed thus far is simpler than other models of disk systems in that we are able to obtain a simple closed-form solution for the response time without relying on iteration and transformation. The results of [Bard81] require the solution of a set of simultaneous equations by the Newton-Raphson method. The method of [Gotl73] for multiple disk drives also requires an iteration of equations to be performed. The use of a machine-repair queueing model [Saat61] for the analysis of the channel queue also yields no closed form solution for the response time. Finally, the results of [Fran74] require Laplace transformations.

E.2 The I/O Submodel for Bulk Requests with Fixed Bulk Size

We now proceed to make a very important extension to the I/O submodel. In a database management system environment such as MDBS where requests are issued in a high-level query language, each request may involve several tracks. Thus, each request to the disk system requires actually the retrieval of several tracks. In the sequel, we assume that requests to the disk system arrive with an exponential inter-arrival distribution at a mean rate L. However, each request is assumed to require the retrieval of T tracks of data which are randomly distributed on the disk tracks. This is a bulk arrival system in the queueing theory terminology. We term T the size of a bulk request. Sometimes, we may also talk of the T subrequests of a bulk request.

The queueing model we use is shown in Figure 21. As in the previous section for single requests, we will consider a drive queue and the channel queue as a composite queue. The composite queue is a queue with bulk arrivals. In order to analyze a queue with bulk arrivals, we need to know the mean and variance of the service time, the mean and variance of the inter-arrival time, and the first and second moments of the size of a bulk request.

The mean and variance of the service time of the composite queue, s and v,

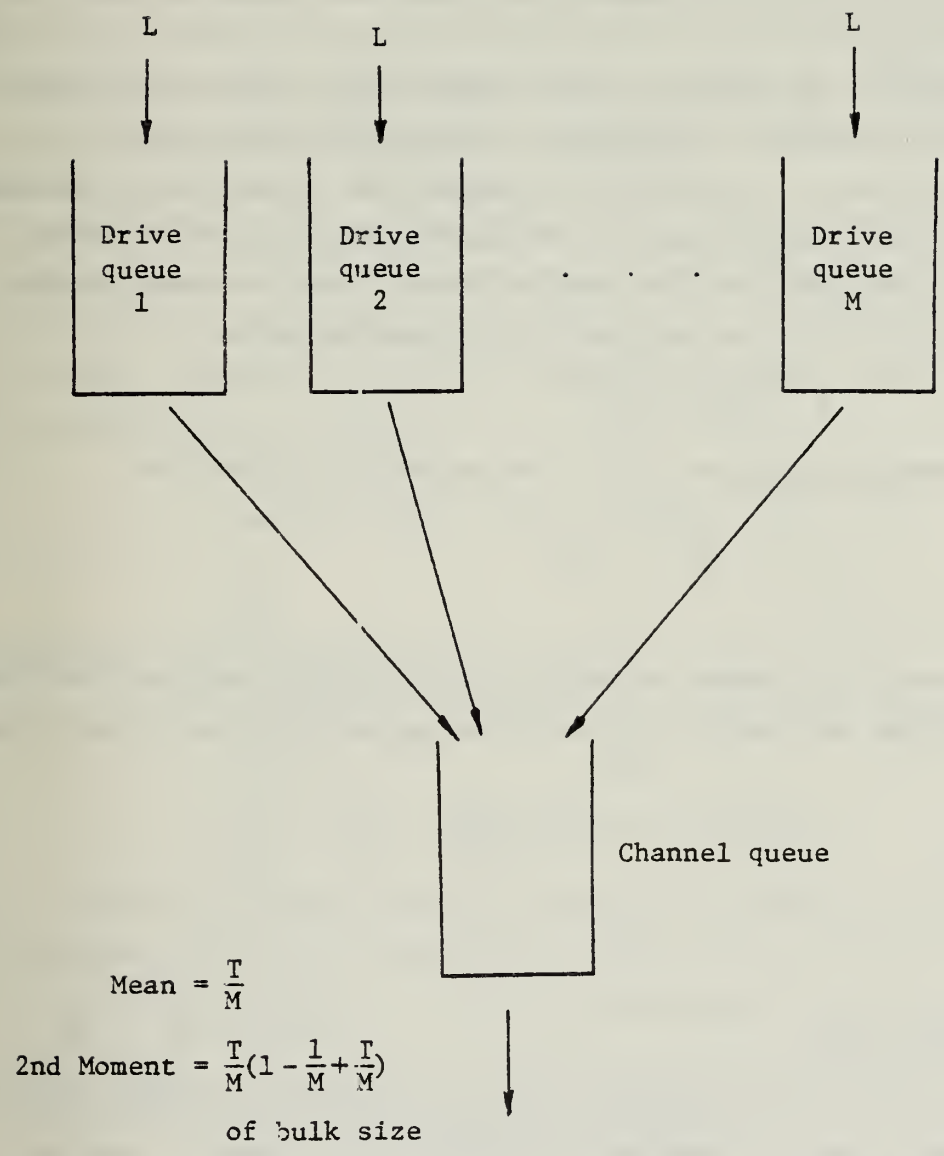


Figure 21. Queueing Model of a Disk System with Bulk Arrivals at Size T at a Rate L

are calculated in the previous section. The only difference is that L in the expressions for s and v must now be replaced by TL.

The mean and variance of the inter-arrival time the composite queue are $\frac{1}{L}$ and $\frac{1}{L^2}$, respectively.

The first moment of the size of a bulk request is $\frac{T}{M}$. The second moment of the size of a bulk request at each module is calculated by assuming a Bernoulli trial as follows. There are T subrequests in a bulk request received by the disk system. For each of these T subrequests, there is a 1/M probability that it will be assigned to a particular drive. Then, the probability that i requests out of T will be assigned to the same module is:

$$\binom{T}{i} (\frac{1}{M})^i (1 - \frac{1}{M})^{T-i}$$

Thus, the second moment of the bulk size of a request at a module is

$$\sum_{i=1}^{T} i^2 \binom{T}{i} (\frac{1}{M})^i (1 - \frac{1}{M})^{T-i}$$

This is simplified to $(T/M)(1 - 1/M + T/M)$.

All the six quantities needed to analyze the composite queue are now known. The total response time may now be calculated using the following formula from [Saat61].

$$r = \frac{s}{2(1-q)} (\frac{t}{r'} + \frac{v}{s^2}) + s$$

where

$q = L*s*r'$,

$r' = T/M$,

$t = (T/M)(1 - 1/M + T/M)$.

Replacing T with 1, we get the same result that we got for the previous analysis when bulk requests were not assumed. Hence, we have reasons to believe that our analysis is correct.

E.3 The I/O Submodel for Bulk Requests with Variable Bulk Size

We now make one final variation in our assumptions for modelling the I/O system of MDBS. In the above analysis, we assumed that the size of a bulk request, as issued to the disks of a back-end, was fixed at T. More generally, a request to MDBS will require the retrieval of T tracks. However, the number of T tracks that will be retrieved at any one back-end varies from 1 to T. Therefore, we need to assume that each disk system (one disk system is asso-

ciated with each back-end) may receive a variable number of subrequests with each bulk request. That is, the size of the bulk requested will vary from 1 to T. If there are n back-ends, then the mean size of the bulk requested at each disk system is T/n and the variance of this bulk size is $(T/n)(1 - 1/n)$. This explains the queueing model shown in Figure 22. The mean and variance of the bulk size of each request as received by each module must now be calculated. This may be calculated as follows.

The probability that there are j subrequests of the bulk request at a drive is equal to the probability that i out of T requests are first chosen to be sent to one of the back-ends multiplied by the probability that j out of these i requests are sent to a particular drive. Then, the mean bulk size at a drive is:

$$= \sum_{i=0}^{T} \binom{T}{i} \left(\frac{1}{n}\right)^i \left(1-\frac{1}{n}\right)^{T-i} \sum_{j=0}^{i} j \binom{i}{j} \left(\frac{1}{M}\right)^j \left(1-\frac{1}{M}\right)^{i-j}$$

$$= \frac{T}{Mn}.$$

Similarly, the second moment of bulk size at a drive is:

$$= \sum_{i=0}^{T} \binom{T}{i} \left(\frac{1}{n}\right)^i \left(1-\frac{1}{n}\right)^{T-i} \sum_{j=0}^{i} j^2 \binom{i}{j} \left(\frac{1}{M}\right)^j \left(1-\frac{1}{M}\right)^{i-j}$$

$$= \frac{T}{Mn}\left(1-\frac{1}{Mn}\right) + \frac{T^2}{M^2n^2}.$$

The final result we are after may now be obtained by starting the equation developed for the bulk request with fixed bulk size. We let $r' = \frac{T}{Mn}$, $t = \frac{T}{Mn}\left(1-\frac{1}{Mn}\right) + \frac{T^2}{M^2n^2}$ and L in the expressions for s and v be replaced by $\frac{LT}{n}$. This is the result we shall use for the response time of the disk system. Thus,

$$r = \frac{qs}{2(1-q)}\left(\frac{t}{r'} + \frac{v}{s^2}\right) + s$$

where

$$q = Lsr'$$

$$r' = \frac{T}{Mn},$$

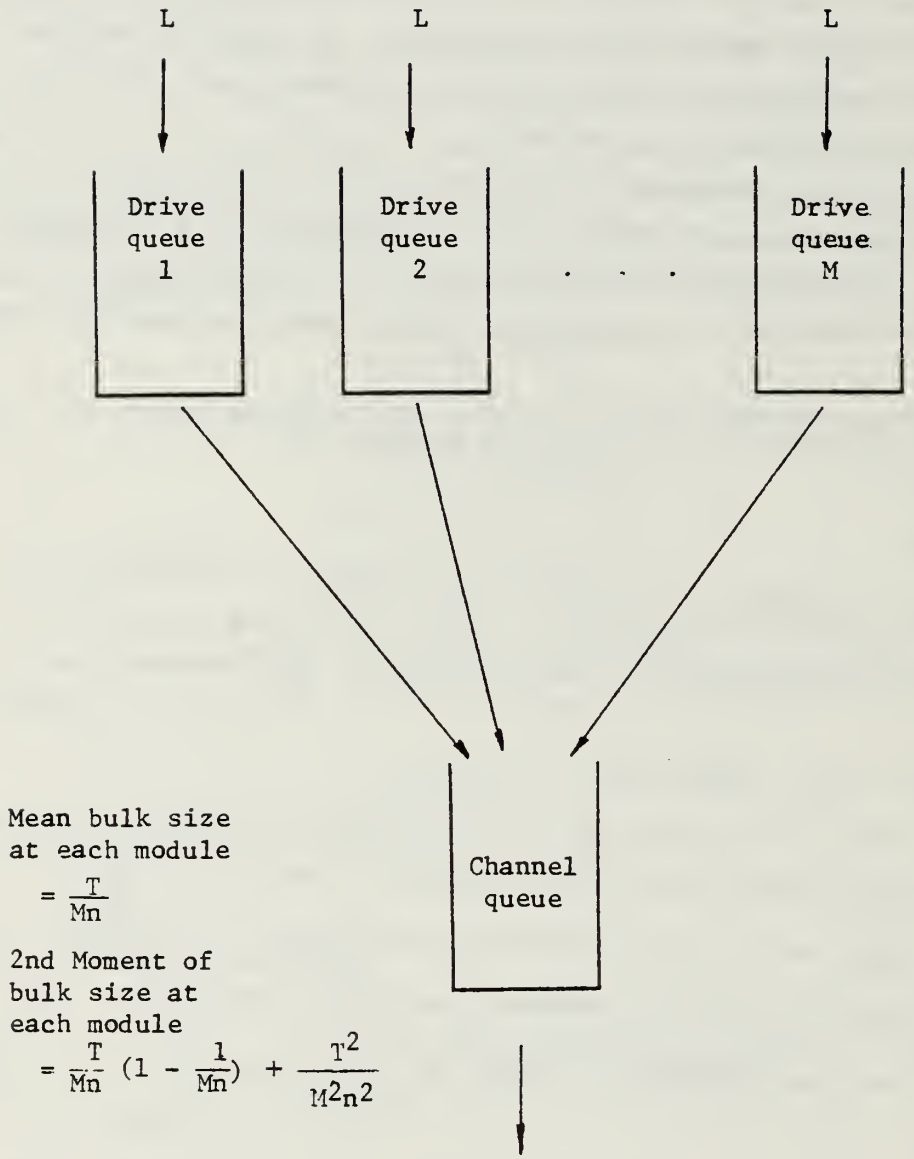
$$t = \frac{T}{Mn}\left(1-\frac{1}{Mn}\right) + \frac{T^2}{M^2n^2}.$$

F. Modelling the Eight Strategies for Evaluation

Having described the I/O submodel thoroughly, we are now ready to describe the overall closed queueing network model of which the I/O submodel is a part. We will develop a separate closed queueing network model for each of the eight

Mean bulk size = $\frac{T}{n}$

variance of bulk size = $\frac{T}{n}(1 - \frac{1}{n})$



Mean bulk size at each module

$= \frac{T}{Mn}$

2nd Moment of bulk size at each module

$= \frac{T}{Mn}(1 - \frac{1}{Mn}) + \frac{T^2}{M^2n^2}$

Figure 22. Queueing Model of a Disk System with Bulk Arrivals of Variable Bulk Size

strategies. For ease in describing these models, we use the following terminology.

tparse	:	time to parse a user request.
tdir	:	time to do descriptor processing in Strategies A, B, C, D and H.
tdire	:	time to do descriptor processing in Strategy E.
tdirf	:	time to do descriptor processing in Strategy F.
tidrg	:	time to do descriptor provessing in Strategy G.
tm	:	time to generate a message.
adgen	:	time for address generation in the controller.
adgen1	:	time for address generation in the back-ends.
T	:	number of tracks to be retrieved for a typical request.
y	:	average number of predicates in a user query.
tbus	:	time to send a user request over the bus.
tbus2	:	time to send retrieved records over bus.
tout	:	time taken by controller to output retrieved records to user.
tproc	:	time taken by a back-end to check a track of records against a user query.

We are now ready to describe the eight models for the eight different strategies.

F.1 The Centralized Model

Consider the sequence of execution of a typical request. The request is first processed at the controller. The controller must parse the request and this will take tparse time units. Next, descriptor processing must be performed on the query in the request. We have already shown how the time for descriptor processing (tdir) may be calculated for the various strategies. Next, the controller must generate the necessary secondary memory addresses using the augmented CDT. This will take adgen time units. Finally, the controller must broadcast the addresses to the back-ends which will take tm time units. Thus, the total time taken to service the request at the controller is (tm + tdir + adgen + tparse). Since each back-end must receive a copy of the request, the controller effectively serves n requests, where the service time of a request is (tm + tdir + tparse + adgen)/n.

The request is now sent over the bus to the back-ends. The bus service time for request processing is tbus. Since n copies of the same request are sent to n respective back-ends, the bus effectively serves n requests. Thus, the service time is really $\frac{tbus}{n}$. The request is now received by the back-ends which use the respective disk systems to retrieve relevant tracks. If the average number of tracks to be retrieved for a request is T, the average number of tracks that has to be retrieved at each back-end is T/n. The service time of a disk system in responding to these T/n I/O requests is obtained from

the I/O submodel described in the previous section.

The retrieved tracks are now being checked against the user's query. It is assumed that tproc msecs have to be spent by a back-end in processing a track of records against a user's query. The records satisfying the query are now sent over the bus to the controller. The service time of the bus in this case is assumed to be tbus2 (to be $\frac{tbus2}{n}$ when there are n requests). Finally, the controller must output the records to the user which takes tout time units. A good approximation of tout is tm, the time to send a message. As before, it is convenient to assume that actually n separate results are sent to the user with an average service time of $\frac{tout}{n}$.

In some cases, some of the devices (i.e., components of MDBS) have more than one queue. For instance, the controller has two queues. The first queue contains requests that must be parsed, processed, and so on. The second queue contains a group of records waiting to be output to the user. Such cases are handled as separate classes in our model. Thus, there are two classes of requests at the controller. Let S(a,b) be the service time for the class b request at device a. Similarly, let V(a,b) be the visit ratio of the class b request at device a which is defined as the number of service completions of class b requests at device a for each service completion from MDBS. A closed queueing network is completely specified when V(a,b) and S(a,b) are specified for all classes and for all devices. Then, the results of [Rood79] may be used to calculate the response times, average queue lengths and utilizations on a per class and per device basis. Let us specify V(a,b) and S(a,b) for our model. Let the controller be device 1, the bus be device 2, a disk system be device 3 and a back-end be device 4.

There are two classes of requests at the controller (device 1). The queue of requests waiting to be parsed, processed, and so on, is designated as class 1 at device 1. The queue of records waiting to be output by the controller to the user is designated as class 2 at device 1. The bus (device 2) also has two classes of requests. The queue of messages from the controller to the back-ends is designated as class 1 at device 2. The queue of records sent from the back-ends to the controller to be output to the user is designated as class 2 at device 2. The disk system (device 3) has only one request class. This is the queue of I/O requests for retrieval of tracks of records at specified addresses which is designated as class 1 at device 3. Finally, each back-end (device 4) has only one request class and this is the queue of records re-

trieved by the disk system which are waiting to be checked against the user's query. This queue is designated as class 1 at device 4. In some of the later models, we will see that a back-end may have two more request classes, making a total of up to three request classes at a back-end. If a model has two or more request classes at a back-end, the following notation is employed. The queue of requests waiting for address generation is designated as class 1 at device 4. The queue of records retrieved by the disk system which are waiting to be checked against the user's query is designated as class 2 at device 4. Finally, the queue of requests waiting for descriptor processing is designated as class 3 at device 4. This completes our discussion of the notation which will be followed for this model and for the models for all the other strategies also. The S and V matrices for the centralized model is as below.

$$S(1,1) = (tdir + tm + tparse + adgen)/n$$
$$S(1,2) = \frac{tout}{n}$$
$$S(2,1) = \frac{tbus}{n}$$
$$S(2,2) = \frac{tbus2}{n}$$
$$S(3,1) = ?$$
$$S(4,1) = tproc$$
$$V(1,1) = n$$
$$V(1,2) = n$$
$$V(2,1) = n$$
$$V(2,2) = n$$
$$V(3,1) = \frac{T}{n}$$
$$V(4,1) = \frac{T}{n}$$

Note that we have put a question mark for S(3,1), the service time of the disk system. The I/O submodel will be used to calculate S(3,1).

F.2 The Partially Centralized Model

The notation we developed for the previous strategy will now be used to explain the remaining models. Let us consider the sequence of execution of a particular request in the partially centralized strategy. First, the controller parses the request and then performs descriptor processing on the request. Finally, it broadcasts the corresponding descriptors to all the back-ends. Thus, the service time for the request at the controller is (tdir + tparse + tm). As before, we assume that there are n requests and that the service time for a request at the controller is $\frac{(tdir + tparse + tm)}{n}$. The request now goes over the bus with a service time of $\frac{tbus}{n}$ to the back-ends.

The back-ends perform address generation taking adgen1 units of time. We use adgen1 rather than adgen as in the previous strategy because the address generation in this strategy and in all the remaining strategies is different from the address generation in the centralized strategy.

Next, the request is presented by a back-end to its disk system for accessing the relevant tracks. The retrieved data are then processed in the back-ends taking a service time of tproc units. The results are then sent back over the bus to the controller. The bus service time is again $\frac{tbus2}{n}$. Finally, the results must be returned to the user and the controller takes $\frac{tout}{n}$ time units for this. The model may be specified by specifying its S and V entries as below.

$$S(1,1) = \frac{(tdir + tparse + tm)}{n}$$

$$S(1,2) = \frac{tout}{n}$$

$$S(2,1) = \frac{tbus}{n}$$

$$S(2,2) = \frac{tbus2}{n}$$

$$S(3,1) = ?$$

$$S(4,1) = adgen1$$

$$S(4,2) = tproc$$

$$V(1,1) = n$$

$$V(1,2) = n$$

$$V(2,1) = n$$

$$V(2,2) = n$$

$$V(3,1) = \frac{T}{n}$$

$$V(4,1) = 1$$

$$V(4,2) = \frac{T}{n}$$

F.3 The Rotating Model

Consider the sequence of execution of a typical request. Two possibilities exist depending on whether the descriptor processing is done at the controller or at one of the back-ends. If the descriptor processing is done at the controller, then the service time at the controller will be $\frac{(tparse + tm + tdir)}{n}$. Otherwise, the service time will be $\frac{(tparse + tm)}{n}$. Since there are two possibilities of request execution in this model as compared to the single request execution possibility in the other models, an additional request class is introduced at the controller for this model and this is designated class 3 at device 1. The request is now broadcast over the bus with a service time of $\frac{tbus}{n}$. If descriptor processing has not already been

done at the controller, then it must be done at the back-ends, and this takes (tdir + tm) units. The additional tm units is needed to communicate the results to all the back-ends. Next, the back-ends perform address generation taking adgenl time units. The disk systems are activated at this time to access tracks. The retrieved data are then processed by the back-ends with a service time of tproc. Records satisfying the request are sent over the bus taking $\frac{tbus2}{n}$ time units and finally, the controller spends $\frac{tout}{n}$ time units outputting the results to the user. The model may be specified by the following service times and visit ratios.

$$S(1,1) = \frac{(tdir + tparse + tm)}{n}$$

$$S(1,2) = \frac{tout}{n}$$

$$S(1,3) = \frac{(tparse + tm)}{n}$$

$$S(2,1) = \frac{tbus}{n}$$

$$S(2,2) = \frac{tbus2}{n}$$

$$S(3,1) = ?$$

$$S(4,1) = adgenl$$

$$S(4,2) = tproc$$

$$S(4,3) = (tdir + tm)$$

$$V(1-1) = \frac{n}{n+1}$$

$$V(1,2) = n$$

$$V(1,3) = \frac{n^2}{n+1}$$

$$V(2,1) = n$$

$$V(2,2) = n$$

$$V(3,1) = \frac{T}{n}$$

$$V(4,1) = 1$$

$$V(4,2) = \frac{T}{n}$$

$$V(4,3) = \frac{1}{n+1}$$

F.4 The Rotating Without Controller Model

Consider the execution sequence of a typical request. The controller does parsing on a request before broadcasting it to all the back-ends. The controller service time is $\frac{(tparse + tm)}{n}$. The request is now broadcast over the bus with a service time of $\frac{tbus}{n}$ and arrives at the back-ends. One of the back-ends will do the descriptor processing for this request. The visit ratio of the descriptor processing queue at each back-end must be adjusted to reflect the

fact that only one out of every n user generated requests will need descriptor processing at that back-end. The back-end that does descriptor processing must then communicate the results to all the back-ends. Hence, the service time of this queue in the back-end is (tdir + tm). Then, all the back-ends will perform address generation independently, taking adgenl time units. Next, the disk system at a back-end will retrieve $\frac{T}{n}$ tracks of data. These tracks of data are checked by the back-ends against the user's query taking tproc units of time per track. The results are shipped over the bus taking $\frac{tbus2}{n}$ time units. Finally, the controller outputs the results to the user taking $\frac{tout}{n}$ time units. The model is entirely specified by specifying the service times and visit ratios as below.

$$S(1,1) = \frac{(tparse + tm)}{n}$$

$$S(1,2) = \frac{tout}{n}$$

$$S(2,1) = \frac{tbus}{n}$$

$$S(2,2) = \frac{tbus2}{n}$$

$$S(3,1) = ?$$

$$S(4,1) = adgenl$$

$$S(4,2) = tproc$$

$$S(4,3) = tdir + tm$$

$$V(1,1) = n$$

$$V(1,2) = n$$

$$V(2,1) = n$$

$$V(2,2) = n$$

$$V(3,1) = \frac{T}{n}$$

$$V(4,1) = 1$$

$$V(4,2) = \frac{T}{n}$$

$$V(4,3) = \frac{1}{n}$$

F.5 The Fully Duplicated Model

This model differs from the previous one only in the way descriptor processing is done. Instead of each back-end doing the descriptor processing for one out of every n requests, each back-end will do a portion of the descriptor processing for every request. The back-ends must then exchange their results. The service times and visit ratios are shown below for this model:

$$S(1,1) = \frac{(tparse + tm)}{n}$$

$$S(1,2) = \frac{tout}{n}$$

$S(2,1) = \frac{tbus}{n}$

$S(2,2) = \frac{tbus2}{n}$

$S(3,1) = ?$

$S(4,1) = adgen1$

$S(4,2) = tproc$

$S(4,3) = \frac{y*tdire}{n} + tm$

$V(1,1) = n$

$V(1,2) = n$

$V(2,1) = n$

$V(2,2) = n$

$V(3,1) = \frac{T}{n}$

$V(4,1) = 1$

$V(4,2) = \frac{T}{n}$

$V(4,3) = 1$

F.6 The Descriptors Dividing by Attribute Model

The only difference between this and the previous model is that tdire is replaced by tdirf in the expression for S(4,3). The value tdirf has been calculated in a previous section and is the time for doing descriptor processing on a single predicate by using Strategy F. Each back-end is expected to handle $\frac{y}{n}$ out of the y predicates in a user query.

F.7 The Descriptors Division Within Attribute Model

Once again, this model is very similar to that for Strategy E and may be obtained from there by setting S(4,3) equal to (y*tdirg + tm). Once again, tdirg is the time for descriptor processing on a single predicate using Strategy G and has been calculated in an earlier section. Note that, unlike the previous strategies, all y of the predicates in a query are handled at each back-end.

F.8 The Fully Replicated Model

Consider the sequence of execution of a typical request. The controller first parses the request and broadcasts it to all the back-ends. Thus, the controller service time is $\frac{(tparse + tm)}{n}$. The requests go over the bus with an average service time of $\frac{tbus}{n}$. The requests are now received at each back-end. Each back-end will first perform descriptor processing and then address generation on a request. Next, $\frac{T}{n}$ track retrieve requests are submitted to the

disk system. The retrieved records are searched against the user's query in the controller and this takes tproc time units for each track of records. The qualified records are now returned via the bus taking a service time of $\frac{tbus2}{n}$ and are then output to the user by the controller taking a service time of $\frac{tout}{n}$. The complete specification of the service times and visit ratio is shown below:

$$S(1,1) = \frac{(tparse + tm)}{n}$$

$$S(1,2) = \frac{tout}{n}$$

$$S(2,1) = \frac{tbus}{n}$$

$$S(2,2) = \frac{tbus2}{n}$$

$$S(3,1) = ?$$

$$S(4,1) = tdir + adgen1$$

$$S(4,2) = tproc$$

$$V(1,1) = n$$

$$V(1,2) = n$$

$$V(2,1) = n$$

$$V(2,2) = n$$

$$V(3,1) = \frac{T}{n}$$

$$V(4,1) = 1$$

$$V(4,2) = \frac{T}{n}$$

This completes our specification of the various models to be employed. The only question to be answered concerns the incorporation of the I/O submodel into the overall models. We shall explain the procedure employed in order to incorporate the I/O submodel into the overall models in the next section.

G. Results of the Queueing Network Modeling of Strategies

In order to solve any of the eight closed queueing network models of MDBS, the service times and visit ratios of all the devices in the network must be specified. We have specified the visit ratios of all the devices including the disk systems. However, the service times of the disk systems have not yet been specified. In this section, we will describe how the service time of a disk system may be calculated.

In Section E.1, we have an expression for the mean service time of the composite queue, in a disk system, in terms of L, the arrival rate of requests to the disk system. However, we did not know then the value of L. Therefore, a technique for calculating L and, hence, s, is developed herein. We use the

following iterative procedure. The following values are used for the various parameters in our calculations.

```
tparse : 20 msecs
tdir   : calculated using equation (2) of Section A
tdire  : calculated using equation (3) of Section A
tdirf  : calculated using equation (4) of Section A
tdirg  : calculated using equation (5) of Section A
tm     : 8 msecs
T      : 18
tbus   : .01 msecs
tbus2  : 16 msecs
tout   : 8 msecs
tproc  : 10 msecs
adgen1 : calculated using equation (1) of Appendix F
adgen  : calculated using equation (2) of Appendix F
```

The number of back-ends is taken from the set {3, 6, 9} and the number of requests in the system is taken from the set {5, 10, 15}.

Assume an initial value for L, the arrival rate of requests to the disk system as follows:

$$\frac{1}{\text{(seek time + rotation time)}}$$

Using this value of L, calculate the value of s, the disk system service time and r, the disk system response time using the expressions derived in Sections E.1 and E.3, respectively. With s, the closed queueing network model is completely specified. It may then be used to calculate utilizations, response times, throughputs and average queue lengths on a per class and per device manner. In particular, the response time of the disk system modelled by the closed queueing network can be calculated. The value of the disk system response time obtained from the closed queueing network model, R is compared to that obtained from the I/O submodel, r. If the two are less than one millisecond apart, the iteration stops. Otherwise, the value of L is modified to be equal to the throughput Tp of the disk system calculated from the closed queueing network model. The iteration procedure is now repeated until the small difference between R and r is obtained. In our experiments with this iteration technique, we found that no more than three iterations were needed. That is, with the chosen initial value of L, the convergence was extremely fast. With the closed queueing network models for the various directory processing strategies being completely specified, we now present the results of our experiments. The results for strategies A to D are presented in Figures 23, 24, 25 and 26, respectively. Since the results for Strategies E and F are identical, they are presented in Figure 27. Finally, the results for Strategies G and H are presented in Figures 28 and 29, respec-

Number of Requests = 5
Number of Back-ends = 3

Utilization	Response Time (msecs)
.3178	36.499
.0336	2.759
.0	0.003
.0672	5.714
.9966	172.960
.0252	1.026

Number of Requests = 5
Number of Back-ends = 6

Utilization	Response Time (msecs)
.5966	26.191
.0630	1.419
.0001	0.002
.1261	3.032
.9506	144.067
.0236	1.023

Number of Requests = 5
Number of Back-ends = 9

Utilization	Response Time (msecs)
.7815	22.251
.0826	0.964
.0001	.0001
.1652	2.102
.8453	119.090
.0206	1.020

Number of Requests = 10
Number of Back-ends = 3

Utilization	Response Time (msecs)
.3189	37.047
.0337	2.760
0.000	0.003
.0674	5.719
1.000	369.856
.0253	1.026

Number of Requests = 10
Number of Back-ends = 6

Utilization	Response Time (msecs)
.6248	32.458
.0660	1.427
.0001	.002
.1320	3.071
.9955	323.830
.0248	1.025

Number of Requests = 10
Number of Back-ends = 9

Utilization	Response Time (msecs)
.8647	39.183
.0914	0.977
.0001	.001
.1827	2.169
.9353	238.085
.0228	1.023

Number of Requests = 15
Number of Back-ends = 3

Utilization	Response Time (msecs)
.3189	37.051
.0337	2.760
.0000	.003
.0674	5.719
1.000	567.697
.0253	1.026

Number of Requests = 15
Number of Back-ends = 6

Utilization	Response Time (msecs)
.6274	33.675
.0663	1.428
.0001	.002
.1326	3.074
.9996	520.885
.0249	1.025

Number of Requests = 15
Number of Back-ends = 9

Utilization	Response Time (msecs)
.8924	53.761
.0943	0.981
.0001	0.001
.1886	2.188
.9652	370.881
.0236	1.024

Figure 23. Queueing Network Model Results for Strategy A

Number of Requests = 5
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1592	14.973
.0336	2.759
.0000	.003
.0672	5.714
.9978	168.301
.1133	30.361
.0252	1.026

Number of Requests = 5
Number of Back-ends = 6

Utilization	Response Time (msecs)
.3019	8.693
.0638	1.419
.0001	.002
.1275	3.028
.9613	134.016
.1469	21.350
.0239	1.023

Number of Requests = 5
Number of Back-ends = 9

Utilization	Response Time (msecs)
.4192	6.526
.0885	0.967
.0001	.001
.1771	2.112
.9063	117.194
.1230	12.507
.0221	1.021

Number of Requests = 10
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1595	15.023
.0337	2.760
.0000	0.003
.0674	5.719
1.0000	365.655
.1136	30.417
.0253	1.026

Number of Requests = 10
Number of Back-ends = 6

Utilization	Response Time (msecs)
.3139	9.193
.0663	1.428
.0001	0.002
.1326	3.074
.9997	325.161
.1527	21.745
.0249	1.025

Number of Requests = 10
Number of Back-ends = 9

Utilization	Response Time (msecs)
.4612	7.714
.0974	0.984
.0001	.001
.1948	2.205
.9970	294.535
.1354	12.849
.0244	1.025

Number of Requests = 15
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1595	15.023
.0337	2.760
.0000	0.003
.0674	5.719
1.000	563.502
.1136	30.417
.0253	1.026

Number of Requests = 15
Number of Back-ends = 6

Utilization	Response Time (msecs)
.3140	9.203
.0663	1.428
.0001	0.002
.1326	3.074
1.0000	526.068
.1528	21.749
.0249	1.026

Number of Requests = 15
Number of Back-ends = 9

Utilization	Response Time (msecs)
.4625	7.826
.0977	0.985
.0001	0.001
.1954	2.209
.9999	497.486
.1358	12.864
.0244	1.025

Figure 24. Queueing Network Model Results for Strategy B

Number of Requests = 5
Number of Back-ends = 3

Utilization	Response Time (msecs)
.0398	13.145
.0336	2.759
.0883	10.226
.0000	0.003
.0673	5.714
.9981	167.974
.1134	30.363
.0252	1.026
.0188	18.218

Number of Requests = 5
Number of Back-ends = 6

Utilization	Response Time (msecs)
.0433	6.582
.0640	1.419
.1919	5.674
.0001	0.002
.1279	3.029
.9645	133.882
.1473	21.358
.0240	1.023
.0204	18.231

Number of Requests = 5
Number of Back-ends = 9

Utilization	Response Time (msecs)
.0423	4.378
.0894	0.967
.2816	4.124
.0001	0.001
.1788	2.115
.9152	117.438
.1243	12.517
.0224	1.021
.0200	18.211

Number of Requests = 10
Number of Back-ends = 3

Utilization	Response Time (msecs)
.0399	13.511
.0337	2.760
.0885	10.239
.0000	.003
.0674	5.719
1.0000	365.406
.1136	30.417
.0253	1.026
.0188	18.222

Number of Requests = 10
Number of Back-ends = 6

Utilization	Response Time (msecs)
.0449	6.609
.0663	1.428
.1989	5.824
.0001	0.002
.1326	3.074
.9999	326.398
.1527	21.747
.0249	1.025
.0212	18.265

Number of Requests = 10
Number of Back-ends = 9

Utilization	Response Time (msecs)
.0462	4.412
.0976	0.985
.3074	4.481
.0001	0.001
.1952	2.207
.9988	300.118
.1356	12.856
.0244	1.025
.0218	18.276

Number of Requests = 15
Number of Back-ends = 3

Utilization	Response Time (msecs)
.0399	13.151
.0337	2.760
.0885	10.239
.0000	.003
.0674	5.719
1.0000	563.253
.1136	30.417
.0253	1.026
.0188	18.222

Number of Requests = 15
Number of Back-ends = 6

Utilization	Response Time (msecs)
.0449	6.610
.0663	1.428
.1990	5.826
.0001	.002
.1326	3.074
1.0000	527.379
.1528	21.749
.0249	1.026
.0212	18.265

Number of Requests = 15
Number of Back-ends = 9

Utilization	Response Time (msecs)
.0463	4.413
.0977	0.985
.3077	4.494
.0001	.001
.1954	2.209
1.0000	564.254
.1358	12.864
.0244	1.025
.0218	18.278

Figure 25. Queueing Network Model Results for Strategy C

Number of Requests = 5
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1177	10.560
.0336	2.759
.0000	.003
.0672	5.714
.9979	167.438
.1133	30.360
.0252	1.026
.0250	18.334

Number of Requests = 5
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2235	5.872
.0638	1.419
.0001	0.002
.1277	3.027
.9627	133.263
.1471	21.348
.0239	1.023
.0238	18.290

Number of Requests = 5
Number of Back-ends = 9

Utilization	Response Time (msecs)
.3118	4.262
.0891	0.967
.0001	0.001
.1782	2.113
.9119	116.841
.1238	12.511
.0223	1.021
.0221	18.247

Number of Requests = 10
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1179	10.581
.0337	2.760
0.0000	0.003
0.0674	5.719
1.0000	364.820
.1136	30.417
.0253	1.026
.0251	18.339

Number of Requests = 10
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2321	6.075
.0663	1.428
.0001	0.002
.1326	3.074
.9998	325.242
.1527	21.746
.0249	1.025
.0247	18.331

Number of Requests = 10
Number of Back-ends = 9

Utilization	Response Time (msecs)
.3414	4.707
.0976	0.985
.0001	0.001
.1951	2.207
.9986	298.250
.1356	12.855
.0244	1.025
.0242	18.321

Number of Requests = 15
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1179	10.581
.0337	2.760
.0000	0.003
.0674	5.719
1.0000	562.667
.1136	30.417
.0253	1.026
.0251	18.339

Number of Requests = 15
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2321	6.077
.0663	1.428
.0001	.002
.1326	3.074
1.0000	526.208
.1528	21.749
.0249	1.026
.0247	18.331

Number of Requests = 15
Number of Back-ends = 9

Utilization	Response Time (msecs)
.3479	4.727
.0977	0.985
.0001	0.001
.1954	2.209
1.0000	502.233
.1358	12.864
.0244	1.025
.0243	18.323

Figure 26. Queueing Network Model Results for Strategy D

Number of Requests = 5
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1176	10.555
.0336	2.758
.0000	0.003
.0672	5.712
.9972	164.720
.1132	30.347
.0252	1.026
.0474	11.842

Number of Requests = 5
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2125	5.747
.0607	1.411
.0001	0.002
.1214	2.994
.9153	115.205
.1398	21.069
.0228	1.021
.1212	17.933

Number of Requests = 5
Number of Back-ends = 9

Utilization	Response Time (msecs)
.2724	4.017
.0778	0.953
.0001	0.001
.1556	2.049
.7966	93.721
.1081	12.254
.0195	1.017
.1296	14.975

Number of Requests = 10
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1179	10.581
.0337	2.760
.0000	0.003
.0674	5.719
1.0000	354.556
.1136	30.417
.0253	1.026
.1009	26.642

Number of Requests = 10
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2314	6.057
.0661	1.427
.0001	.002
.1322	3.070
.9967	273.279
.1523	21.712
.0248	1.025
.1974	29.718

Number of Requests = 10
Number of Back-ends = 9

Utilization	Response Time (msecs)
.3325	4.575
.0950	0.980
.0001	.001
.1900	2.180
.9725	213.057
.1320	12.760
.0238	1.024
.2209	23.666

Number of Requests = 15
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1179	10.581
.0337	2.760
0.0000	0.003
.0674	5.719
1.0000	541.824
.1136	30.417
.0253	1.026
.1676	47.798

Number of Requests = 15
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2321	6.076
.0663	1.428
.0001	.002
.1326	3.074
.9998	446.030
.1527	21.748
.0249	1.025
.2637	43.193

Number of Requests = 15
Number of Back-ends = 9

Utilization	Response Time (msecs)
.3407	4.702
.0973	0.984
.0001	0.001
.1947	2.205
.9963	362.853
.1353	12.849
.0243	1.025
.2905	33.565

Figure 27. Queueing Network Model Results for Strategies E and F

Number of Requests = 5
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1175	10.548
.0336	2.758
.0000	0.003
.0671	5.711
.9960	161.248
.1131	30.238
.0252	1.025
.0750	19.302

Number of Requests = 5
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2103	5.727
.0601	1.410
.0001	0.002
.1202	2.988
.9061	112.809
.1384	21.024
.0225	1.021
.1343	20.315

Number of Requests = 5
Number of Back-ends = 9

Utilization	Response Time (msecs)
.2545	3.933
.0727	0.948
.0001	0.001
.1454	2.025
.7442	86.998
.1010	12.160
.0182	1.016
.1625	20.696

Number of Requests = 10
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1179	10.581
.0337	2.760
.0000	0.003
.0674	5.719
1.0000	358.209
.1136	30.417
.0253	1.026
.0753	19.334

Number of Requests = 10
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2318	6.066
.0662	1.428
.0001	.002
.1325	3.072
.9986	289.951
.1526	21.730
.0248	1.025
.1480	20.972

Number of Requests = 10
Number of Back-ends = 9

Utilization	Response Time (msecs)
.3335	4.585
.0953	0.980
.0001	0.001
.1906	2.183
.9753	216.819
.1324	12.768
.0238	1.024
.2129	22.545

Number of Requests = 15
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1179	10.581
.0337	2.760
0.0000	.003
.0674	5.719
1.0000	556.056
.1136	30.417
.0253	1.026
.0753	19.334

Number of Requests = 15
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2321	6.077
.0663	1.428
.0001	0.002
.1326	3.074
1.0000	490.343
.1528	21.749
.0249	1.026
.1482	20.989

Number of Requests = 15
Number of Back-ends = 9

Utilization	Response Time (msecs)
.3417	4.720
.0976	0.985
.0001	0.001
.1952	2.208
.9992	409.053
.1357	12.860
.0244	1.025
.2182	22.857

Figure 28. Queueing Network Model Results for Strategy G

Number of Requests = 5
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1175	10.552
.0336	2.758
0.0000	0.003
.0671	5.712
.9963	164.279
.1546	43.423
.0252	1.026

Number of Requests = 5
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2148	5.781
.0614	1.413
.0001	0.002
.1228	3.003
.9256	120.384
.2172	35.150
.0230	1.022

Number of Requests = 5
Number of Back-ends = 9

Utilization	Response Time (msecs)
.2769	4.053
.0791	0.955
.0001	0.001
.1582	2.058
.8097	97.088
.2076	25.509
.0198	1.018

Number of Requests = 10
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1179	10.581
.0337	2.760
0.0000	0.003
.0674	5.719
1.0000	361.281
.1552	43.608
.0253	1.026

Number of Requests = 10
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2318	6.068
.0662	1.428
.0001	0.002
.1325	3.073
.9988	301.412
.2344	36.928
.0248	1.025

Number of Requests = 10
Number of Back-ends = 9

Utilization	Response Time (msecs)
.3374	4.639
.0964	0.982
.0001	0.001
.1928	2.194
.9868	244.849
.2530	27.925
.0241	1.024

Number of Requests = 15
Number of Back-ends = 3

Utilization	Response Time (msecs)
.1179	10.581
.0337	2.760
0.0000	0.003
.0674	5.719
1.0000	559.128
.1552	43.608
.0253	1.026

Number of Requests = 15
Number of Back-ends = 6

Utilization	Response Time (msecs)
.2321	6.077
.0663	1.428
.0001	0.002
.1326	3.074
1.0000	501.585
.2347	36.983
.0269	1.026

Number of Requests = 15
Number of Back-ends = 9

Utilization	Response Time (msecs)
.3418	4.724
.0977	0.985
.0001	0.001
.1953	2.209
.9997	442.489
.2563	28.224
.0244	1.025

Figure 29. Queueing Network Model Results for Strategy H

tively. Each of these figures contains two columns, one for the utilization and one for the response time. These two measures are recorded on a per-class-and-per-device basis. The order of presentation of the results within a figure for a particular strategy is the same as the order of presentation of the service time and visit ratio in the description of the model for that Strategy. For instance, in the description of the model for Strategy A, we first presented the service time and visit ratio for class 1 at device 1. Thus, in Figure 23, the first row gives the utilization and response time for class 1 at device 1. In reading these figures, we should keep in mind that device 1 is the controller, device 2 is the bus, device 3 is a disk system and device 4 is a back-end.

We also present a comparative set of results for the eight different strategies in Figure 30. Each entry in this figure represents the overall response time of MDBS using one of these eight strategies. Most of the discussion that follows is based on the results shown in this figure. However, in order to explain some of the results shown in this figure, we will have to refer back to the results presented in Figures 23 through 29.

As expected, the centralized strategy (i.e., Strategy A) is the worst of the lot. It is consistently the worst strategy over the entire range of values for the number of requests and the number of back-ends. There are situations when the response time obtained by use of the centralized strategy is almost 50% higher than when using some of the other strategies. For example, when the number of back-ends is taken as nine and the number of requests is fifteen, the response time of the centralized strategy is 1255.16 msecs. On the other hand, the response time using Strategy E, the fully duplicated strategy, is only 810 msecs. The reason for the poor performance of the centralized strategy is, as we have expected, owing to the fact that the controller is becoming a bottleneck. For the particular numbers of requests and numbers of back-ends under consideration, the use of Strategy E will result in a mere 35% utilization of the controller. This figure is obtained by adding the first two entries in the utilization column of Figure 27. On the other hand, the use of the centralized strategy lead to a controller utilization as high as 90% which is obtained by adding the first two entries in the utilization column of Figure 23.

The partially centralized strategy (i.e., Strategy B) was consistently better than the centralized strategy. The response time improvement of the

RESPONSE TIME TABLES

(in msecs)

Number of requests = 5

Strategies / # of back-ends	A	B	C	D	E	F	G	H
3	1173.7067	1111.5368	1106.0955	1093.1209	1076.7827	1076.7826	1055.9013	1087.1987
6	617.087	503.2648	490.2142	484.078	428.6259	428.6259	421.2351	458.5127
9	467.0594	334.3705	315.5591	313.2972	263.8941	263.8941	249.3356	284.3019

Number of requests = 10

Strategies / # of back-ends	A	B	C	D	E	F	G	H
3	2356.7506	2295.8904	2288.7749	2277.5511	2215.9679	2215.9679	2237.8885	2269.5116
6	1194.2616	1080.4319	1068.6451	1061.9707	905.9084	905.9084	956.0146	1005.6126
9	858.1617	701.0841	682.5620	681.4713	509.5228	509.5228	517.1716	588.9918

Number of requests = 15

Strategies / # of back-ends	A	B	C	D	E	F	G	H
3	3543.8065	3482.9716	3475.8566	3464.6326	3339.5784	3339.5784	3424.9681	3456.5894
6	1792.7510	1683.2213	1671.5949	1664.8886	1424.3426	1424.3426	1557.2938	1607.0621
9	1255.1678	1108.0571	1090.8745	1089.6635	810.6118	810.6118	903.2192	985.4986

Figure 30. MDBS Response Times Under Various Directory Management Strategies

partially centralized strategy over the centralized strategy is most evident when the number of back-ends is very large. For example, when the number of back-ends is nine and the number of requests is five, the centralized strategy has a response time of 467.06 msecs and the partially centralized strategy has a response time of 334.37 msecs which is an improvement of about 40%. The reason that the improvement is more visible at larger number of back-ends is because this is precisely when the controller in the centralized strategy becomes highly utilized. In other words, the partially centralized strategy is a more extensible strategy than the centralized strategy. Comparing corresponding entries in Figures 23 and 24, we see that the controller utilization under the partially centralized strategy is 1/2 the controller utilization under the centralized strategy.

Next, let us consider the results of using the rotating strategy, (i.e., Strategy C). As we recall, the rotating strategy is an improvement over the partially centralized strategy because the descriptor processing is shared by the back-ends and the controller, instead of being done entirely in the controller. The results bear out the fact that the rotating strategy is indeed superior to the partially centralized strategy, since it consistently outperforms the partially centralized strategy, in terms of response time, over the entire range of values for the number of back-ends and the number of requests. The improvements are not dramatic. The improvement in average response time is in the order of ten or twenty milliseconds. The largest improvement occurs when the number of back-ends is the largest. This implies, of course, that the rotating strategy is more extensible than the partially centralized strategy and, hence, more preferable to us as designers of an extensible system.

Next, let us consider the results of using the rotating without controller strategy (Strategy D). The rationale of using this strategy was that it might provide some improvement over the rotating strategy because of the fact that it serves to alleviate the controller limitation problem to a greater degree. This is because the controller is no longer involved in descriptor processing. The results of Figure 30 bear this out to some degree. In fact, the rotating without controller strategy provides a better response time than the rotating strategy over the entire range of values for the number of requests and the number of back-ends. However, the improvement of this strategy over the rotating strategy is only marginal. Thus, it is seen that the improvement pro-

vided by this strategy over the rotating one is never more than about ten milliseconds. Furthermore, the percentage improvement in response time remains fairly constant over the entire range of values of the two parameters being varied. Thus, the percentage improvement is (1104.09 - 1093.12)/1104.09, when the number of back-ends is three and the number of requests is five. For the same number of requests and six back-ends, the percentage improvement is (490.2142 - 484.0788)/490.2142. Both these represent an improvement of about 11%. A look at the controller utilizations in Figures 25 and 26 tells us that the controller is indeed a little less utilized in the rotating without controller strategy. For instance, the controller utilization when the number of back-ends is three and the number of requests is five for the rotating strategy is 16.17%. On the other hand, the controller utilization in the rotating without controller strategy for the same values of number of back-ends and number of requests is 15.13%.

Let us now consider the results of the fully duplicated strategy (Strategy E). As we recall, this strategy, along with the next two strategies, are the three strategies that employ parallel descriptor processing during the descriptors search phase. Our results indicate that such a strategy is likely to be better than all the strategies considered thus far. Comparing it with the best strategy considered so far (the rotating without controller strategy), we see that the fully duplicated strategy can lead to as much as 34% lower response time. This happens when the number of back-ends is nine and the number of requests is 15. For these particular number of back-ends and number of requests, the response time using the fully duplicated strategy is 810.6118 msecs whereas the response time using the rotating without controller strategy is 1089.6635 msecs. The results also indicate that the improvement of the fully duplicated strategy over the rotating without controller strategy becomes more evident at larger number of back-ends and larger number of requests. In other words, the fully duplicated strategy is more extensible than the other strategies we have considered so far.

The utilization of the disk system plays an important part in the disparity in the response time between these two strategies. The utilization of the disks in the rotating without controller strategy (the fifth row of each table in Figure 26) is a little higher than in the fully duplicated strategy (See the fifth row of each table in Figure 27). However, a small increase in the disk utilization can increase the overall response time by a large amount owing

to the fact that the disk service time is large. This is one of the causes of the better response time in the fully duplicated strategy. Another reason for the better response time is the fact that the descriptor search phase now takes a shorter length of time. For instance, when the number of back-ends is nine and the number of requests is five, the descriptor search time in the rotating without controller strategy is 18.247 msecs (see last row of corresponding table in Figure 26) while it is only 14.975 msecs (see last row of corresponding table in Figure 27) in the fully duplicated strategy. This was the reason why we had expected the fully duplicated strategy to perform better than the rotating without controller strategy. The fact that the disk system response time is also improved is unexpected.

The utilization of the controller is also seen to be a little better in the case of the fully duplicated strategy. Thus, when the number of back-ends is nine and the number of requests is five, the total controller utilization is 40.09% if the rotating without controller strategy is used, while it is only 35.02% if the fully duplicated Strategy is used. This also leads to some improvement in total response time.

The results for the descriptors division by attribute strategy (Strategy F) are exactly the same as for the fully duplicated strategy (Strategy E).

Next, let us consider the results for the descriptors division within attribute strategy (Strategy G). We recall that this strategy also employs parallel descriptor processing. For small number of requests, this strategy performs marginally better than the fully duplicated strategy. For large number of requests, however, the fully duplicated strategy outperforms the descriptors division within attribute strategy. Furthermore, the fully duplicated strategy outperforms the descriptors division within attribute strategy the most, when the number of back-ends is the largest. Thus, when the number of back-ends is nine and the number of requests is fifteen, the response time under the fully duplicated strategy is 810.62 msecs and the response time under the descriptors division within attribute strategy is 903.2192 msecs. This is an improvement of 10.3%.

Finally, let us consider the results of the fully replicated strategy (Strategy H). The results indicate that it is inferior to the three strategies which employ parallel descriptor processing and superior the the other four strategies.

Consider, once again, the results of Figure 30 for the fully duplicated

strategy (Strategy E). When the number of requests is five and the number of back-ends is three, the response time is 1076.78 msecs. When the number of back-ends is increased to six and the number of requests is kept unchanged, the response time improves to 428.63 msecs. That is, when the number of back-ends increases to twice of its original number, the response time of MDBS improves better than one half of its original time. This is a surprising result. Our data placement strategy only ensures us that the doubling of the number of back-ends may halve the number of tracks to be retrieved at each back-end. Thus, we have expected that, in the best case, a doubling of the number of back-ends would lead to a halving of the response time. Hence, the results of Figure 30 are better than expected. An examination of the results in Figure 27 reveals to us the reason for the better than expected improvement in response time. Once again, it was the response time of the disk system that played an important role in this unexpected result. When the number of back-ends is increased as indicated, it turns out that the disk utilization decreases from 99.72% to 91.53%. As a result, the disk system response time decreases from 164.72 msecs per request to 115.21 msecs per request. Thus, not only is the number of tracks to be retrieved decreased, so is the time to retrieve each track. This explains the greater-than-expected decrease in response time. Similar reasons account for the fact that when the number of back-ends is tripled to nine while keeping the number of requests constant at five, the response time is shortened to 263.8941 msecs which is 24.5% of the original time of 1076.7827 msecs (we have expected that the response time can be no better than 33% of what it was when the number of back-ends was three). In other words, by choosing such a strategy for directory management in MDBS, we expect that an increase in the number of back-ends in MDBS by a factor of n will cause an improvement in response time which will be better than a factor of n.

In conclusion, the results indicate that the fully duplicated strategy (Strategy E) and the descriptors division by attribute strategy (Strategy F) are the best strategies over a wide range of values for the number of requests and the number of back-ends. The choice is now between these two strategies. The advantage of the latter strategy is that it would occupy less storage space, since descriptors are not duplicated. However, in the following section, we will study the difference between these two strategies in terms of storage requirements for typical number of directory attributes and number of

descriptors per attribute. Furthermore, in another section, we will examine whether the duplication of descriptors, as in the fully duplicated strategy, is necessary for efficient handling of update requests.

4.2.4 Storage Requirements of Directory Management Strategies

In this section, we shall compare the eight strategies on the basis of their storage requirements. As we have already indicated, two types of directory based tables are necessary in MDBS. The first type of table is the descriptor-to-descriptor-id table (DDIT) and the second type of table is the augmented cluster definition table (CDT). We shall discuss for each strategy the amount of storage needed for both types of tables, in turn.

Primarily, the eight different directory management strategies differ in the amount of storage needed for storing descriptor-to-descriptor-id tables. Except for the centralized strategy, there is no difference among the different strategies in the amount of storage needed for storing the augmented cluster definition tables. Since the size of the augmented CDTs is much larger than the size of the descriptor-to-descriptor-id tables, it is expected that there will be no significant difference among the different strategies in terms of storage requirements. Let us examine them more carefully.

A. Size Estimation of the Descriptor-to-Descriptor-Id Tables (DDITs)

We assume that a database has i directory attributes and that there are D descriptors per attribute. Let k be the number of descriptors that can be stored in a page of b bytes. All strategies, except strategies F and G, require the following amount of storage for storing the descriptor tables.

$$S = u(\frac{D}{k})bi \text{ bytes}$$

where, u(a) stands for the nearest integer equal to or greater than a. In Strategies A and B, DDITs are stored entirely at the controller. Thus, the size of the tables is S bytes. In Strategy C, these tables are duplicated in the controller and at all the back-ends. Thus, a total of (n+1)S bytes are required by these tables. In Strategies D, E and H, the tables are duplicated in all the back-ends. Hence, in these strategies, the total storage required by the tables is nS bytes. In Strategy F, the total space occupied by the descriptor tables is:

$$u(\frac{D}{k})bn\,u(\frac{i}{n}) \text{ bytes} .$$

Finally, for Strategy G, the storage needed is

$$u(\frac{D}{nk})bin \text{ bytes.}$$

B. Size Estimation of the Augmented Cluster Definition Table (CDT)

We recall that logically the augmented CDT consists of cluster definitions and corresponding secondary memory addresses. We assume that the augmented CDT is physically implemented as follows. Consider a database with i directory attributes and D descriptors per attribute. Then the augmented CDT consists of iD entries. Each entry is formed with a descriptor id followed by a list of cluster definitions and their corresponding secondary addresses. Now, the number of bytes needed to represent a cluster is

$$u(\frac{lgD}{8})i.$$

Here, lg stands for the logarithm of base two. In the ensuing discussion, we will assume that t bytes are needed to store a track address and that p bytes are needed to indicate a back-end number. Finally, we assume that the average cluster size is c tracks, where $c \geq 1$. Thus, the size, X, of each CDT entry for the centralized strategy is given by

$$u(\frac{lgD}{8}) + D^{i-1}(u(\frac{lgD}{8})i + c(t+p))$$

Thus, the total size of the augmented CDT in the centralized strategy is given by iDX.

In the case of every other strategy, the size of an augmented CDT entry is

$$u(\frac{lgD}{8}) + \frac{cD^{i-1}}{n}(u(\frac{lgD}{8})i + t), \text{ if } c \leq n;$$

$$u(\frac{lgD}{8}) + D^{i-1}(u(\frac{lgD}{8})i + \frac{ct}{n}), \text{ otherwise.}$$

If we denote the size of an augmented CDT entry for all strategies except the centralized one as Y, the total size of CDT is niDY for all strategies except the centralized one.

The following table shows the results of our study for the combined size of both tables. In the sequel, we shall refer to the combined size of the DDITs and the augmented CDT as the directory size.

STRATEGY	DIRECTORY SIZE
A	$S + iDX$
B	$S + niDY$
C	$(n+1)S + niDY$
D	$nS + niDY$
E	$nS + niDY$
F	$u(\frac{D}{k})bn\ u(\frac{i}{n}) + niDY$
G	$u(\frac{D}{nk})bin + niDY$
H	$nS + niDY$

C. Interpretation of the Results on Sizes

Let us now proceed to perform some actual calculations on the directory sizes. We let i, the number of directory attributes, be 5, b, the page size, be 512 bytes, k, the number of descriptors per page, be 85, t, the number of bytes in a track address, be 4; and p, the number of bytes to represent a back-end number, be 1. The number of back-ends is taken from the set {2, 5, 8}, the cluster size c is taken from the {1, 2, 3, 4}, and the number of descriptors D is taken from the set {10, 15, 20}.

The results are shown in Figure 31. The results in this figure show the directory sizes in K bytes for various number of descriptors per attribute, size of a cluster and number of back-ends.

Strategy A has the smallest storage requirements when the size of a cluster is large. However, it has the largest storage requirements when the size of a cluster is small. Because of its very large worst case storage requirements, Strategy A is not preferable for implementation in MDBS. Let us now consider the storage requirements for the remaining seven strategies. It is clear that there is no significant difference in the storage requirements for these seven strategies. In fact, the largest difference in storage requirements among these seven strategies is .05%.

Since there is no 'superior' strategy in terms of storage requirements, the strategy chosen for implementation in MDBS should be the one which is found to be superior in terms of performance. Our results have shown that Strategies E and F were the superior ones in terms of performance. In the following section, it will be seen that the duplication of descriptors, as in Strategy E,

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 10
CLUSTER SIZE = 1

Strategies / Backends	A	B	C	D	E	F	G	H
2	5002.6	4502.7	4507.8	4505.2	4505.2	4503.2	4502.2	4502.2
5	5002.6	4502.8	4515.6	4513.1	4513.1	4502.8	4513.1	4513.1
8	5002.6	4503.0	4523.4	4520.9	4520.9	4504.5	4520.9	4520.9

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 10
CLUSTER SIZE = 2

Strategies / Backends	A	B	C	D	E	F	G	H
2	7502.6	9002.7	9007.8	9005.2	9005.2	9003.2	9005.2	9005.2
5	7502.6	9002.8	9015.6	9013.1	9013.1	9002.8	9013.1	9013.1
8	7502.6	9003.0	9023.4	9020.9	9020.9	9004.5	9020.9	9020.9

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 10
CLUSTER SIZE = 3

Strategies / Backends	A	B	C	D	E	F	G	H
2	10002.6	11002.7	11007.8	11005.2	11005.2	11003.2	11005.2	11005.2
5	10002.6	13502.8	13515.6	13513.1	13513.1	13502.8	13513.1	13513.1
8	10002.6	13503.0	13523.4	13520.9	13520.9	13504.5	13520.9	13520.9

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 10
CLUSTER SIZE = 4

Strategies / Backends	A	B	C	D	E	F	G	H
2	12502.6	13002.7	13007.8	13005.2	13005.2	13003.2	13005.2	13005.2
5	12502.6	18002.8	18015.6	18013.0	18013.0	18002.8	18013.0	18013.0
8	12502.6	18003.0	18023.4	18020.9	18020.9	18004.5	18020.9	18020.9

Figure 31. Directory Size (in kbytes) for Various Directory Management Strategies

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 15
CLUSTER SIZE = 1

Strategies / Back-ends	A	B	C	D	E	F	G	H
2	37971.4	34174.6	34179.7	34177.1	34177.1	34175.1	34177.1	34177.1
5	37971.4	34174.8	34187.6	34185.0	34185.0	34174.8	34185.0	34185.0
8	37971.4	34175.0	34195.5	34193.0	34193.0	34176.6	34193.0	34193.0

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 15
CLUSTER SIZE = 2

Strategies / Back-ends	A	B	C	D	E	F	G	H
2	56955.8	68346.5	68351.6	68349.0	68349.0	68347.0	68349.0	68349.0
5	56955.8	68346.7	68359.5	68456.9	68356.9	68346.7	68356.9	68356.9
8	56955.8	68346.9	68367.4	68364.8	68364.8	68348.4	68364.8	68364.8

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 15
CLUSTER SIZE = 3

Strategies / Back-ends	A	B	C	D	E	F	G	H
2	75940.1	83534.0	83539.1	83536.5	83536.5	83534.5	83536.5	83536.5
5	75940.1	102518.6	102531.4	102528.8	102528.8	102518.6	102528.8	102528.8
8	75940.1	102518.8	102539.3	102536.7	102536.7	102520.3	102536.7	102536.7

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 15
CLUSTER SIZE = 4

Strategies / Back-ends	A	B	C	D	E	F	G	H
2	94924.5	98721.5	98726.6	98724.0	98724.0	98722.0	98724.0	98724.0
5	94924.5	136690.4	136703.2	136700.7	136700.7	136690.4	136700.7	136700.7
8	94924.5	136690.7	136711.1	136708.6	136708.6	136692.2	136708.6	136708.6

Figure 3I. (Contd.)

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 20
CLUSTER SIZE = 1

Back-ends \ Strategies	A	B	C	D	E	F	G	H
2	160002.7	144002.8	144007.9	144005.3	144005.3	144003.3	144005.3	144005.3
5	160002.7	144003.1	144015.9	144013.3	144013.3	144003.1	144013.3	144013.3
8	160002.7	144003.4	144023.8	144021.3	144021.3	144004.9	144021.3	144021.3

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 20
CLUSTER SIZE = 2

Back-ends \ Strategies	A	B	C	D	E	F	G	H
2	240002.7	288002.8	288007.9	288005.3	288005.3	288003.3	288005.3	288005.3
5	240002.7	288003.1	288015.9	288013.3	288013.3	288003.1	288013.3	288013.3
8	240002.7	288003.4	288023.8	288021.3	288021.3	288004.9	288021.3	288021.3

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 20
CLUSTER SIZE = 3

Back-ends \ Strategies	A	B	C	D	E	F	G	H
2	320002.7	352002.8	352007.9	352005.3	352005.3	352003.3	352005.3	352005.3
5	320002.7	432003.1	432015.9	432013.3	432013.3	432003.1	432013.3	432013.3
8	320002.7	432003.4	432023.8	432021.3	432021.3	432004.9	432021.3	432021.3

NUMBER OF DESCRIPTORS PER ATTRIBUTE = 20
CLUSTER SIZE = 4

Back-ends \ Strategies	A	B	C	D	E	F	G	H
2	400002.7	416002.8	416007.9	416005.3	416005.3	416003.3	416005.3	416005.3
5	400002.7	576003.1	576015.9	576013.3	576013.3	576003.1	576013.3	576013.3
8	400002.7	576003.4	576023.8	576021.3	576021.3	576004.9	576021.3	576021.3

Figure 31. (Contd.)

is necessary for efficient handling of some kinds of update requests. Thus, we choose to adopt Strategy E for directory management in MDBS.

4.3 The Entire Process of Request Execution

In the previous section, we described the process of directory management in MDBS. In this final section, we shall discuss the entire sequence of actions performed by MDBS in processing the four different types of requests. We shall discuss each type of request, in turn.

4.3.1 Executing a Retrieve Request

We recall that the syntax of a retrieve request in MDBS is as follows.

RETRIEVE Query Target-list [BY Attribute][WITH Pointer]

The sequence of actions taken by MDBS in order to execute a retrieve has already been described in some detail in an earlier section. We shall repeat some of the discussion here, for completeness.

The controller will first parse the request and determine that it is a retrieve request. Next, the controller will broadcast the request to all the back-ends. The back-ends will perform descriptor processing and address generation under Strategy E as described in the previous section. Upon completion, each back-end has a list of secondary memory addresses of the tracks which contain the relevant records. These tracks are accessed by the back-end. The query in the request is used to select the records from these tracks. First, the records satisfying the query are selected. If a BY-clause is specified in the retrieve request, the selected records are grouped by the values of the attribute in the BY-clause. If no BY-clause is specified in the retrieve request, all the selected records are treated as a single group. Next, for each group of selected records, the values of all attributes in the target-list are extracted from the records of the group. If no aggregate operator is specified on an attribute in the target-list, the extracted values of the group are returned to the controller. If an aggregate operator is specified on an attribute in the target-list, some computation is performed on all the attribute values in the records of the group and a single aggregate value is returned to the controller. This completes the actions performed by a back-end on each group of selected records. If a WITH-clause is specified in the retrieve request, the secondary memory addresses of all selected records must

also be sent to the controller by each back-end..

The controller will wait for responses from all the back-ends. Upon receiving all the responses (i.e., attribute values, aggregate values or addresses) from all back-ends, the controller will forward these responses to the user that issued the retrieve request. This completes the execution of the retrieve request.

4.3.2 Executing a Delete Request

As we recall, the syntax of a delete request is

DELETE Query

The execution of this request in MDBS is similar to the execution of a retrieve request. The controller will first parse the request and determine that it is a delete request. Next, the controller will broadcast the request to all back-ends. The back-ends will perform descriptor processing and address generation under Strategy E. Upon completion, each back-end has a list of secondary memory addresses of tracks which contain relevant records. Records of these tracks are retrieved from the secondary memory by respective back-ends. The query in the delete request is used to select the records which are to be deleted. These selected records are then marked for deletion. The track space occupied by the marked records is not immediately recovered. Such recovery of space will be done during database reorganization time (see Chapter 7). After the records are marked, the marked records are written back to the same tracks by each back-end. If all the records in a track are marked for deletion, the address of this track is removed from all entries in which it appears in the augmented cluster definition table (CDT). Finally, each back-end will send an acknowledgement to the controller to indicate that it has finished executing the delete request. Upon receiving the acknowledgements from all the back-ends, the controller will inform the user or user program that the delete request has successfully been completed.

4.3.3. Executing an Update Request

The syntax of an update request in MDBS is as follows

UPDATE Query Modifier

We recall that the modifier in an update request specifies the new value to be taken by the attribute being modified and that it may be one of the types

described below.

Type-0 : <attribute = constant>
Type-I : <attribute = f(attribute)>
Type-II : <attribute = f(attribute-1)>
Type-III : <attribute = f(attribute-1) of Query>
Type-IV : <attribute = f(attribute-1) of Pointer>

In the simplest case, a modifier indicates the new value to be taken by the attribute being modified (i.e., type-0). In the more involved cases, the modifiers specify the new value to be taken by the attribute being modified as a function f of the 'old' value of that attribute (i.e., type-I) or values of some other attribute of the record to be updated (e,g., types II, III or IV). The other attribute is called the base attribute (i.e.,attribute-1 in the specification). We will first describe the execution of an update request containing modifiers of types 0, I and II. We will then describe the execution of an update request containing modifiers of type-III or IV.

An update request containing a modifier of types 0, I or II is broadcast by the controller to all the back-ends. The back-ends will perform descriptor processing and address generation under Strategy E. Afterwards, each back-end has a list of secondary memory addresses of the tracks containing the relevant records. These tracks are accessed by respective back-ends and the records satisfying the query are selected from these tracks. These are the records to be updated.

Each of these records is updated using the modifier in the update request. If the modifier is of type-0, the new value to be taken by the attribute being modified in a record to be updated is provided in the modifier. If the modifier is of type-I, the new value to be taken by the attribute (being modified) in a record (to be updated) is computed as a function (specified in the modifier) of the value of the same attribute. Finally, if the modifier is of type-II, the new value to be taken by the attribute (being modified) in a record (to be updated) is computed as a function f (specified in the modifier) of the value of the base attribute in that record.

Due to its change in attribute values, an updated record may remain in the same cluster to which it (more precisely, its pre-updated version) belonged or it may now belong to a different cluster. In the latter case, a record is said to change cluster. Recall that a cluster is a group of records such that every record in the cluster is derived from the same set of descriptors. Thus, an updated record will belong to a different cluster only if the set of descriptors from which it is derived is different from the set

of descriptors from which the pre-updated version was derived. If the attribute being modified in an updated record is not a directory attribute, the updated record continues to be derived from the same set of descriptors, since only directory attributes affect the descriptors. Hence, the updated record does not change cluster. If the attribute being modified is a directory attribute, an updated record may change cluster. Consider, for example, a database with the descriptors as depicted in Figure 14. Consider also the following record

(<Location,NAPA>, <Number,32123>, <Balance,50>).

This record is clearly derived from the descriptor set {D2, D3, D5} (see Figure 14 again). After updating the value of the Balance which is a directory attribute in this record, the updated record is now shown below.

(<Location,NAPA>, <Number,32123>, <Balance,100>)

By the way, this update uses type 0 modifier. The updated record is also derived from the same descriptor set {D2, D3, D5}. Hence, the record does not change cluster. On the other hand, if the value of the Balance is changed to 600, the newly updated record is not derived from the descriptor set {D2, D3, D5}. Instead, it is derived from the set {D2, D4, D5}. Hence, the record changes cluster.

In order to check whether or not a newly updated record changes cluster, it is necessary for a back-end to search the descriptor-to-descriptor-id table (DDIT). To facilitate such search, we have decided that each back-end should replicate the descriptors for all the directory attributes in its secondary memory. This is an additional reason that we decided on Strategy E. For example, if Strategy F were employed, a back-end would need to access descriptors placed at other back-ends in order to determine clusters of updated records. Consequently, bus traffic will be excessive which would create the so-called control message problem.

Finally, each back-end will send an acknowledgement to the controller to indicate that it has finished processing the update request. When it has received acknowledgements from all back-ends, the controller will output a message to the user to signal successful completion of the update request. This completes the processing of an update request containing modifiers of type-0, I or II.

Now, let us describe the execution of an update request containing a type-III or IV modifier. In this case, another record must first be retrieved

by MDBS on the basis of a user-provided query or pointer. After the record is retrieved, the controller will extract the base attribute value v from the retrieved record. It will then compute the function f (specified in the type-III or IV modifier) on the value v and thus obtain a new value v'. The controller will then form a type-0 modifier of the form

<a = v'>

where a is the attribute being modified, i.e., the same attribute appeared to the left of the equality sign in the type-III or IV modifier. The original type-III or IV modifier in the update request is now replaced with this newly created type-0 modifier. In other words, MDBS converts an update request containing a type-III or IV modifier to an update request containing a type-0 modifier. This update request containing a type-0 modifier may now be executed in the same manner described previously.

4.3.4 Executing an Insert Request

The syntax of an insert request in MDBS is

INSERT Record

The controller will first parse the request and determine that it is an insert request. Next, the controller will broadcast the request to all the back-ends. The back-ends will perform descriptor processing under Strategy E. The descriptor search phase under Strategy E for record insertion is more specialized than the descriptor search phase under the same strategy for other requests. For example, for the retrieve request, the descriptor search phase involves parallel descriptor processing of multiple predicates of a given query conjunction. For the insert request, the descriptor search phase merely involves parallel descriptor processing of multiple attribute-value pairs (i.e., keywords) of the record to be inserted. The specialized descriptor processing for record insertion tends to simplify the processing effort. More specifically, if the record for insertion contains x directory attributes, each back-end will determine the corresponding descriptors for $\frac{x}{n}$ attribute-value pairs. At the end of the descriptor search phase, the single cluster to which the record to be inserted is known to the back-end(s) whose secondary memory(memories) has(have) been accommodating the cluster. The reason that more than one back-end may be involved in accommodating the cluster in consideration is that the cluster being sufficiently large has been evenly distributed by the data placement strategy over several back-ends' secondary

memories at the database-creation time. Consequently, MDBS must decide which back-end's secondary memory is to be used for accommodating the new record.

The address generation phase for record insertion requires an additional step under Strategy E. Instead of generating the secondary memory address immediately upon the completion of descriptor processing, each back-end sends at most one cluster id of the corresponding descriptors to the controller. By consulting a cluster-id-to-next-back-end table (CINBT), the controller can select the secondary memory of a specific back-end for record insertion. The CINBT which is depicted in Figure 32 is created for the controller at the database-creation time by the data placement strategy. Upon receiving a message from the controller, the specific back-end will then continue into the address generation phase by producing a secondary address for record insertion.

Thus, in this chapter, we have completely described the execution of the four types of requests that may be issued to MDBS.

This table is stored at the controller to be used during record insert and database re-organization.

Cluster Id	Next Back-end for Insertion	Number of Tracks*
.	.	.
.	.	.
.	.	.

* Information in this and additional columns will be used for database re-organization. (See Chapter 7 for re-organization)

Figure 32. The Cluster-Id-To-Next-Back-end Table (CINBT)

REFERENCES

[Adabyy] ADABAS Reference Manual, Software AG, Reston, Virginia.

[Astr75] Astrahan, M.M., and Chamberlin, D.D., "Implementation of a Structured English Query Language," Communications of the ACM, Vol. 18, No. 10, October 1975, pp. 580-587.

[Astr76] Astrahan, M.M., et. al., "System R: Relational Approach to Database Management," ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, pp. 189-222.

[Auer80] Auer, H., "RDBM - A Relational Data Base Machine," Technical Report No. 8005, University of Braunschweig, June 1980.

[Babb79] Babb, E., "Implementing a Relational Database by Means of Specialized Hardware," ACM Transactions on Database Systems, Vol. 4, No. 1, March 1979, pp. 1-29.

[Bane77] Banerjee, J., Hsiao, D.K. and Kerr, D.S., "DBC Software Requirements for Supporting Network Databases," Technical Report, OSU-CISRC-TR-77-4, The Ohio State University, Columbus, Ohio, June 1977.

[Bane78] Banerjee, J. and Hsiao, D.,K., "Concepts and Capabilities of a Database Computer," ACM Transactions on Database Systems, Vol. 3, No. 4, December 1978, pp. 347-384. Also available in Baum, R.I., Hsiao, D.K. and Kannan, K., "The Architecture of a Database Computer --Part I: Concepts and Capabilities," Technical Report OSU-CISRC-TR-76-1, The Ohio State University, Columbus, Ohio, September 1976.

[Bane79] Banerjee, J., Hsiao, D.K. and Kannan, K., "DBC - A Database Computer for Very Large Databases," IEEE Transactions on Computers, Vol. C-28, No. 6, June 1979, pp. 414-429.

[Bane80] Banerjee, J., Hsiao, D.K. and Ng, F., "Database Transformation, Query Translation and Performance Analysis of a Database Computer in Supporting Hierarchical Database Management," IEEE Transactions on Software Engineering, March 1980; Also available in Hsiao, D.K., Kerr, D.S. and Ng, F.K., "DBC Software Requirements for Supporting Hierarchical Databases," Technical Report OSU-CISRC-TR-77-1, The Ohio State University, Columbus, Ohio, April 1977.

[Bard81] Bard, Y., "A Model of Shared DASD and Multipathing," CACM, Vol. 23, No. 10, October 1980, pp. 564-572.

[Baye76] Bayer, R., and Metzger, J.K., "On the Encipherment of Search Trees and Random Access Files," ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, pp. 37-52.

[Baye80] Bayer, R., et al, "Parallelism and Recovery in Database Systems," ACM Transactions on Database Systems, Vol. 5, No. 2, June 1980.

[Bent75] Bentley, J.L., "Multidimensional Binary Search Trees Used for Associative Searching," CACM, September 1975, Vol. 18, No. 9, pp. 509-517.

[Bora80] Boral, H., et al, "Parallel Algorithms for the Execution of Relational Database Operations," Computer Sciences Technical Report No. 402, University of Wisconsin-Madison, October 1980.

[Bora81] Boral, H. and DeWitt, D.J., "Processor Allocation Strategies for Multiprocessor Database Machines," ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, pp. 227-265.

[Cana74] Canaday, R.H., et al, "A Back-End Computer for Data Base Management," Communications of the ACM, Vol. 17, No. 10, October 1974, pp. 575-582.

[Card75] Cardenas, A.F., "Analysis and Performance of Inverted Data Base Structures," Communications of the ACM, Vol. 18, No. 5, May 1975, pp. 253-263.

[Cham74] Chamberlin, D.D. and Boyce, R.F., "A Deadlock-Free Scheme for Resource Locking in a Database Environment," IFIP, August 1974, pp. 340-343.

[Cham75] Chamberlin, D.D., Gray, J.J. and Traiger, I.L., "Views, Authorization and Locking in a Relational Data Base System," Proceedings of the National Computer Conference, 1975, pp. 425-430.

[Chan80a] Chang, J.M. and Fu, K.S., "A Dynamic Clustering Technique for Physical Database Design," Proceedings of the ACM SIGMOD Conference on Management of Data, Santa Monica, California, May 14-16, 1980.

[Chan80b] Chang, C.C., Lee, R.C.T. and Du, H.C., "Some Properties of Cartesian Product Files," Proceedings of the ACM SIGMOD Conference on Management of Data, Santa Monica, California, May 14-16, 1980.

[Chu 78] Chu, W.W., Lee, D. and Iffla, B., "A Distributed Processing System for Naval Data Communication Networks," AFIPS Conference Proceedings, Vol. 47, 1978, pp. 783-793.

[Chu 80] Chu, W.W., et al, "Task Allocation in Distributed Data Processing," Computer Magazine, Vol. 13, No. 11, November 1980, pp. 57-69.

[Come79] Comer, D., "The Ubiquitous B-Tree," ACM Computing Surveys, Vol. 11, No. 2, June 1979.

[Cope73] Copeland, C.P., Lipovski, G.J. and Su, S.Y.W., "The Architecture of CASSM: A Cellular System for Non-Numeric Processing," Proceedings of the First Annual Symposium on Computer Architecture, December 1973, pp. 121-128.

[Cosm75] Cosmetalos, G.P., "Approximate Explicit Formula for the Average Queueing Time in the Processes M/D/r and D/M/r," Information, Vol. 13, October 1975.

[Datayy] Data Management System (DMS 1100), Amercian National Standard COBOL Data Manipulation Language, Programmer Reference UP-7908, Sperry Univac Computer Systems, St. Paul, Minn.

[Date75] Date, C.J., "An Introduction to Data Base Systems," Addison-Wesley, Reading, Mass., 1975.

[Denn78] Denning, P.J. and Buzen, J.P., "The Operational Analysis of Queueing Network Models," Computing Surveys, Vol. 10, No. 3, September 1978, pp. 225-261.

[Dewi78] DeWitt, D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Data Base Management Systems," Proceedings of the Fifth Annual Symposium on Computer Architecture, 1978.

[Dewi80] DeWitt, D.J., and Wilkinson, K.K., "Database Concurrency Control in Local Broadcast Networks," Computer Sciences Technical Report 396, University of Wisconsin-Madison, August 1980.

[Down77] Downs, D. and Popek, G.J., "A Kernel Design for a Secure Data Base Management System," Proceedings of the Conference on Very Large Databases, Tokyo, Japan, October 6-8, 1977, pp. 507-514.

[Epst78] Epstein, R., Stonebraker, M. and Wong, E., "Distributed Query Processing in a Relational Data Base System," Memorandum No. UCB/ERL M78/18, Electronics Research Laboratory, University of California, Berkeley, April 1978.

[Epst80] Epstein, R. and Hawthorn, P., "Design Decisions for the Intelligent Database Machine," Proceedings of the National Computer Conference, AFIPS, Vol. 49, 1980, pp. 237-241.

[Eswa76] Eswaran, K.P., et al, "The Notions of Consistency and Predicate Locks in a Database System," CACM, Vol. 19, No. 11, November 1976, pp. 624-633.

[Fern75] Fernandez, E.B., et al, "An Authorization Model for a Shared Data Base," Proceedings of the ACM-SIGMOD Conference on Management of Data, San Jose, California, pp. 23-31, May 1975.

[Fran74] Franklin, M.A. and Sen, A., "An Analytic Response Time Model for Single and Dual-Density Disk Systems," IEEE Transactions on Computers, Vol. C-23, No. 12, December 1974.

[Fran77] Franta, W.R., "The Process View of Simulation," Computer Science Library, North Holland, 1977.

[Gard77] Gardarin, G. and Lebeux, P., "Scheduling Algorithms for Avoiding Inconsistency in Large Databases," Third International Conference on VLDB, Japan, October, 1977.

[Good80] Goodman, J.R. and Despain, A.M., "A Study of the Interconnection of Multiple Processors in a Data Base Environment," International Conference on Parallel Processing, 1980, pp. 269-278.

[Gotl73] Goelieb, C.C. and Macewen, G.H., "Performance of Movable-Head Disk Storage Devices," JACM, Vol. 20, No. 4, October 1973, pp. 604-623.

[Gray78] Gray, J., "Notes on Data Base Operating Systems," Operating Systems, Lecture Notes in Computer Science, Vol. 60, Springer-Verlag, 1978.

[Grif76] Griffiths, P.P. and Wade, B.W., "An Authorization Mechanism for a Relational Database System," ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, pp. 242-255.

[Hawt79] Hawthorn, P. and Stonebraker, M., "Performance Analysis of a Relational Database Management System,", Proceedings of the ACM SIGMOD Conference on Management of Data, Boston, May 30 - June 1, 1979.

[Hawt80] Hawthorn, P. and DeWitt, D.J., "Performance Analysis of Alternative Database Machine Architectures," Computer Sciences Technical Report No. 383, University of Wisconsin-Madison, March 1980.

[Hawt81] Hawthorn, P., "The Effect of Target Applications on the Design of Database Machines," Proceedings of the ACM SIGMOD Conference on Mangement of Data, April 29 - May 1, 1981, pp. 188-197, Ann Arbor, Michigan.

[Hsia70] Hsiao, D.K. and Harary, F.A., "A Formal System for Information Retrieval from Files," Communications of the ACM, Vol. 13, No. 2, February 1970, pp. 67-73.

[Hsia79a] Hsiao, D.K., Kerr, D.S. and Madnick, S.E., "Computer Security, Problems and Solutions", Academic Press, 1979.

[Hsia79b] Hsiao, D.K., Kerr, D.S. and Nee, C.J., "Database Access Control in the Presence of Context Dependent Protection Requirements," IEEE Transactions on Software Engineering, Vol. SE-5, No. 4, July 1979.

[Hsia80] Hsiao, D.K., "Design Issues of High-Level Language Database Computers," Proceedings of the International Workshop on High-Level Language Computer Architecture, May 26-28, 1980, pp. 92-98, Fort Lauderdale.

[Hsia81] Hsiao, D.K., "A Survey of Concurrency Control Mechanisms for Centralized and Distributed Databases," Technical Report, OSU-CISRC-TR-81-1, The Ohio State University, Columbus, Ohio, February 1981.

[Idmsyy] IDMS, Concepts and Facilities, Cullinane Corporation, Wellesley, Mass.

[Jenn77] Jenny, C.J., "Process Partitioning in Distributed Systems," Digest of Papers NTC 1977, 1977.

[Jord81] Jordan, J.J. Banerjee, J. and Batman, R., "Precision Locks," Proceedings of the ACM SIGMOD Conference on Management of Data, April 29 - May 1, 1981, pp. 143-147.

[Kann77a] Kannan, K., "The Design and Performance of a Database Computer," Ph.D. Dissertation, Ohio State University, 1977.

[Kann77b] Kannan, K., Hsiao, D.K. and Kerr, D.S., "A Microprogrammed Keyword Transformation Unit for a Database Computer," Proceedings of the Tenth Annual Workshop on Microprogramming, October 1977, Niagara Falls, New York, pp. 71-79.

[Kann78] Kannan, K., "The Design of a Mass Memory for a Database Computer," Proceedings of the Fifth Annual Symposium on Computer Architecture, April 1978, Palo Alto, California, pp. 44-50.

[Katz80] Katz, R.H., "Database Design and Translation for Multiple Data Models," University of California, Berkeley, Ph.D. Thesis.

[Klei75] Kleinrock, L., "Queueing Systems," Vol. I and II, John Wiley, 1975.

[Knut75] Knuth, D., "The Art of Computer Programming," Vol. III, Addison-Wesley, Reading, Mass., 1975, pp. 475-476.

[Litw78] Litwin, W., "Virtual Hashing : A Dynamically Changing Hashing," Fourth International Conference on VLDB, Berlin, September 1978.

[Liu 75] Liu, M.T. and Reames, C.C., "A Loop Network for the Simultaneous Transmission of Variable Length Messages," Proceedings of the Second Annual Conference on Computer Architecture, January 1975, pp. 7-12.

[Lowe76] Lowenthal, E.I., "The Backend Computer, Part I and Part II," Auerbach (Data Management) Series, 24-01-04 and 24-01-05, 1976.

[Mary76] Maryanski, F.J., Fisher, P.S. and Wallentine, V.E., "A User-Transparent Mechanism for the Distribution of a CODASYL Data Base Management System," Technical Report, TR CS 76-22, Kansas State University, December 1976.

[Mary77] Maryanski, F.J., "Performance of Multi-processor Backend Database Systems," Conference on Information Sciences and Systems, August 1977, pp. 437-441.

[Mary80] Maryanski, F.J., "Backend Database Systems," Computing Surveys, Vol. 12 No. 1, March 1980, pp. 3-25.

[Metc76] Metcalfe, R.M. and Boggs, D.R., "Ethernet: Distributed Packet Switching for Local Computer Networks," CACM, Vol. 19, pp. 395-404, July 1976.

[McCa75] McCauley, E.J., "A Model for Data Secure Systems," Ph.D. Dissertation, Ohio State University, Columbus, Ohio, 1975.

[Meno80] Menon, M.J. and Hsiao, D.K., "The Impact of Auxilliary Information and Update Operations on Database Computer Architecture," International Congress on Applied Systems Research and Cybernetics, December 12-16, 1980.

[Miss80] Missikoff, M. and Terranova, M., "An Overview of the Project DBMAC for a Relational Database Machine," unpublished technical report, IASI-CNR, personal communication.

[Moha81] Mohan, C., Fussell, D., and Silberschatz, A., "Concurrency, Compatibility, and Commutativity in non-two-phase Locking Protocols," Unpublished, Department of Computer Sciences, University of Texas, Austin.

[Reis79] Reiser, M., "Mean Value Analysis of Queueing Networks, A New Look at an Old Problem," Performance of Computer Systems, North-Holland, 1979.

[Rive76] Rivest, R.L., "Partial-Match Retrieval Algorithms," SIAM Journal of Computing, Vol. 5, No. 1, March 1976, pp. 19-50.

[Rood79] Roode, J.D., "Multiclass Operational Analysis of Queueing Networks," Performance of Computer Systems, North-Holland, 1979.

[Rose77a] Rosenthal, R.S., "An Evaluation of Data Base Management Machines," Annual Computer Related Information System Symposium, U.S. Air Force Academy, 1977.

[Rose77b] Rosenthal, R.S., "The Data Management Machine, a Classification," Workshop on Computer Architecture for Non-Numeric Processing, May 1977, pp. 35-39.

[Roth74] Rothnie, J.R. and Lozano, T., "Attribute Based File Organization in a Paged Memory Environment," CACM, Vol. 17, No. 2, February 1974, pp. 63-69.

[Roth80] Rothnie, J.R., et al, "Introduction to a System for Distributed Databases (SDD-1)," ACM Transactions on Database Systems, Vol. 5, No. 1, March 1980, pp. 1-17.

[Saat61] Saaty, L., "Elements of Queueing Theory with Applications," McGraw-Hill, New York, 1961.

[Sama80] Samari, N.K. and Schneider, M.G., "A Queueing Theory Based Analytical Model of a Distributed Computer Network," IEEE Transactions on Computers, Vol. C-29, No. 11, Nov. 1980.

[Sava76] Savage, J.E., "The Complexity of Computing," John Wiley, New York, 1976.

[Schn73] Schneiderman, B., "Optimum Database Reorganization Points," CACM, Vol. 16, No. 6, June 1973, pp. 362-365.

[Schu79] Schuster, S.A., Nguyen, H.G., Ozkarahan, E.A, and Smith, K.C., "RAP.2 - An Associative Processor for Databases and its Applications," IEEE Transactions on Computers, Vol. C-28, No. 6, June 1979, pp. 446-458.

[Song80] Song, S.W., "A Highly Concurrent Tree Machine for Database Application," International Conference on Parallel Processing, 1980, pp. 259-268.

[Stea76] Stearns, R.E., Lewis, P.M. II and Rosenkrantz, D.J., "Concurrency Control for Database Systems," Proceedings 17th IEEE Symposium on Foundations of Computer Science, October 1976, pp. 19-32.

[Ston74] Stonebraker, M.R. and Wong, E., "Access Control in a Relational Data Base Management System by Query Modification," Proceedings of ACM Annual Conference, San Diego, California, November 1974, pp. 180-187.

[Ston76a] Stonebraker, M. and Neuhold, E., "A Distributed Data Base Version of INGRES," Proceedings of the Second Berkeley Workshop on Distributed Data Bases and Computer Networks, Berkeley, California, May 1976.

[Ston76b] Stonebraker, M. and Rubinstein, P., "The INGRES Protection System," ACM Annual Conference, 1976, pp. 80-84

[Ston78] Stonebraker, M., "A Distributed Data Base Machine," Memorandum No. UCB/ERL M78/23, Electronics Research Laboratory, University of California, Berkeley, May, 1978.

[Ston79] Stonebraker, M., "Muffin: A Distributed Data Base Machine," First International Conference on Distributed Computing Systems, 1979, pp. 459-469.

[Tane81] Tanenbaum, A.S., "Computer Networks," Prentice-Hall Inc., 1981.

[Systyy] SYSTEM 2000 Reference Manual, MRI Systems Corporation, Austin, Texas.

[Thom79] Thomas, R.H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," ACM Transactions on Database Systems, Vol. 4, No. 2, June 1979.

[Totayy] TOTAL Reference Manual, Cincom Systems Corporation, Cincinnati, Ohio.

[Wah 80] Wah, B.W. and Yao, B.S., "DIALOG - A Distributed Processor Organization for Database Machine," Proceedings of the National Computer Conference, 1980, pp. 243-253.

[Wong71] Wong, E. and Chiang, T.C., "Canonical Structure in Attribute Based File Organization," CACM, Vol. 14, No. 9, September 1971, pp. 593-597.

APPENDIX A: FORMAL SPECIFICATION OF DML

The following is the BNF syntax for the DML of Chapter 3. Square brackets [ ] are used to indicate optional constructs.

```
Predicate                 :=  (attribute rel-op value)
attribute                 :=  char-string
attribute-being-modified  :=  attribute
base-attribute            :=  attribute
rel-op                    :=  <|≤|>|≥|=|≠
value                     :=  string|number|float
Conjunct                  :=  (Predicate)|(Conjunct ∧ Predicate)
Query                     :=  (Conjunct)|(Query ∨ Conjunct)
Stat                      :=  AVG|MAX|MIN|SUM|COUNT
list-el                   :=  Stat (attribute)
list                      :=  attribute|list-el|list,attribute|list,list-el
Target-list               :=  (list)
Attrib-val-pair           :=  <attribute,value>
Half-record               :=  Attrib-val-pair|Half-record, Attrib-val-pair
Record                    :=  (Half-record)
Pointer                   :=  number
Modifier                  :=  type-0|type-I|type-II|type-III|type-IV
type-0                    :=  <attribute-being-modified=value>
type-I                    :=  <attribute-being-modified=expr1>
type-II                   :=  <attribute-being-modified=expr2>
type-III                  :=  <attribute-being-modified=expr2 of Query>
type-IV                   :=  <attribute-being-modified=expr2 of Pointer>
Request                   :=  Insert|Delete|Update|Retrieve
Insert                    :=  INSERT Record
Delete                    :=  DELETE Query
Update                    :=  UPDATE Query Modifier
Retrieve                  :=  RETRIEVE Query Target-list [BY Attribute]
                              [WITH Pointer]
uc-letter                 :=  A|B|C|....|Z
string                    :=  uc-letter|uc-letter string
lc-letter                 :=  a|b|c|....|z
char-string               :=  uc-letter|char-string lc-letter
digit                     :=  0|1|2|3|4|5|6|7|8|9
number                    :=  digit|digit number
float                     :=  number. number
add-op                    :=  +|-
mult-op                   :=  *|÷
expr1                     :=  arith-term1|expr1 add-op arith-term1
arith-term1               :=  arith-factor1|arith-term1 mult-op arith-factor1
arith-factor1             :=  attribute-being-modified|number
expr2                     :=  arith-term2|expr2 add-op arith-term2
arith-term2               :=  arith-factor2|arith-term2 mult-op arith-factor2
arith-factor2             :=  base-attribute|number
```

APPENDIX B: PROOF THAT A RECORD BELONGS TO ONE AND ONLY ONE CLUSTER

Consider a record R consisting of the directory keywords K_1, K_2,... K_n, where K_i contains directory attribute A_i. If n is zero, then the record R contains no directory keywords. Hence, it is derived from the cluster defined by the empty set of descriptors and this is the only cluster it is derived from. Next, let us consider the case when n is non-zero. For purposes of contradiction, let a record R belong to two different clusters X and Y. Consider directory keyword K_1 in R. Since R belongs to cluster X, X must contain a descriptor X_1, from which K_1 is derived. Similarly, X must contain a descriptor X_2 from which K_2 is derived. Therefore, X is defined by the descriptor set $\{X_1, X_2, \ldots X_n\}$. Similarly, Y is defined by the descriptor set $\{Y_1, Y_2, \ldots, Y_n\}$. Now, by the definition of the X_is and the Y_is, the following n statements must be true.

(1) K_1 is derived from both X_1 and Y_1.
(2) K_2 is derived from both X_2 and Y_2.
⋮
(n) K_n is derived from both X_n and Y_n.

Using the statement (1) above, we shall show that $X_1 = Y_1$.

The first point to be noted is that X_1 and Y_1 must both contain A_1, since, K_1 contains A_1. Let K_1 be of the form $<A_1,V>$. Now, there are three possibilities for X_1 as below.

(a) X_1 is a type-A descriptor of the form $(L_1 \leq A \leq U_1)$.
(b) X_1 is a type-B descriptor of the form $(A_1 = V_1)$.
(c) X_1 is a type-C descriptor of the form A_1.

Similarly, there are three possibilities for Y_1 as below.

(a) Y_1 is a type-A descriptor of the form $(L_2 \leq A_1 \leq U_2)$.
(b) Y_1 is a type-B descriptor of the form $(A_1 = V_2)$.
(c) Y_1 is a type-C descriptor of the form A_1.

Thus, there are nine possibilities for the combination of X_1 and Y_1. We shall consider all of the nine cases, in turn.

Case 1: X_1 is $(L_1 \leq A_1 \leq U_1)$.
Y_1 is $(L_2 \leq A_2 \leq U_2)$.

Now, since K_1 is derived from X_1, V lies between L_1 and U_1. Also, since K_1 is derived from Y_1, V lies between L_2 and U_2. Rule 1 of the rules for forming descriptors (see Chapter 3) states that the ranges specified in type-A descriptors for a given attribute must be mutually exclusive. Since the ranges of X_1 and Y_1 are not mutually exclusive, they cannot be two different descriptors. Hence, they must be the same descriptor. Thus, $X_1 = Y_1$.

Case 2: X_1 is $(L_1 \leq A_1 \leq U_1)$
Y_1 is $(A_1 = V_2)$.

Since K_1 is derived from X_1, V lies between L_1 and U_1. Since K_1 is is derived from Y_1, V_1 must be equal to V_2. Thus, V_2 lies between L_1 and U_1. However, this is in violation of Rule 2 for forming descriptors, since, there is a type-B descriptor whose value is enclosed in the range of a type-A descriptor. Hence, it is not possible that X_1 is of type-A and Y_1 is of type-B.

Case 3: X_1 is $(L_1 \leq A_1 \leq U_1)$.
Y_1 is A_1.

This is a clear violation of Rule 3 for forming descriptors which states that an attribute that appears in a type-C descriptor cannot appear also in a type-A or type-B descriptor. Thus, this case is not possible.

Case 4: X_1 is $(A_1 = V_1)$.
Y_1 is $(L_2 \leq A_1 \leq U_2)$.

Since K_1 is derived from X_1, V must be equal to V_1. However, since K_1 is also derived from Y_1, V, and hence V_1, must lie between L_2 and U_2. This is a violation of Rule 2 for forming descriptors. Hence, this case is not possible.

Case 5: X_1 is $(A_1 = V_1)$.
Y_1 is $(A_1 = V_2)$.

Since K_1 is derived from X_1, V must be equal to V_1. But, since K_1 is derived from Y_1 also, V is also equal to V_2. Hence, $V_1 = V_2$. Hence, $X_1 = Y_1$.

Case 6: X_1 is $(A_1 = V_1)$
Y_1 is A_1.

This is a clear violation of Rule 3 for forming descriptors since an attribute that appears in a type-C descriptor cannot also appear in in a type-A descriptor. Hence, this case is not possible.

Case 7: X_1 is A_1.
Y_1 is $(L_2 \leq A_1 \leq U_2)$.

This is again a violation of Rule 3 for forming descriptors and, hence, this case is not possible.

Case 8: X_1 is A_1.
Y_1 is $(A_1 = V_2)$.

Once again, we have a violation of Rule 3 for forming descriptors. Hence, this case is not possible.

Case 9: X_1 is A_1.
Y_1 is A_1.

Clearly, $X_1 = Y_1$.

Out of all the nine cases considered, only three were found to be possible. In each of these three cases, X_1 was found to be equal to Y_1. Thus, $X_1 = Y_1$.

We may similarly use statement (2) to show that X_2 is equal to Y_2, statement (3) to show that X_3 is equal to Y_3, and so on. Thus, $X = Y$. Hence, R does not belong to two different clusters. Hence, a record can belong to one and only one cluster.

APPENDIX C: DIRECTORY MANAGEMENT ALGORITHMS

Here, we will describe two algorithms. The first algorithm determines the cluster to which a record for insertion belongs and it also determines the secondary memory addresses of this cluster. This algorithm is called the directory management algorithm for an insert request. The algorithm is presented as two different procedures because it proceeds in two distinct phases called the descriptor search phase and the address generation phase.

The second algorithm describes how the corresponding set of clusters (and their addresses) for the query in a non-insert request are determined. This algorithm is called the directory management algorithm for a non-insert request. Once again, the algorithm is presented as two different procedures.

Three tables are used in these algorithms. The first table is called the pointer table (PT). It has as many entries as there are directory attributes. Each entry in the PT consists of two fields. The first field contains a directory attribute and the second field contains a pointer to another table called the descriptor-to-descriptor-id table (DDIT). The pointer in the second field of an entry in PT points to the location in the DDIT where the descriptors of the attribute in the first field of the PT entry are stored.

The DDIT has as many entries as there are descriptors. Each entry consists of two fields. The first field contains a descriptor and the second field contains a descriptor id. A view of a DDIT is shown in Figure 14.

Finally, there is a table called the cluster definition table (CDT). There are as many entries in the CDT as there are clusters in the database. Each entry in the CDT consists of three fields. These contain a cluster number, a cluster definition (as a descriptor set) and one or more secondary memory addresses, respectively. The secondary memory addresses in the third field are of the cluster in the first field. A view of a CDT is shown in Figure 15.
Physically, we assume that the CDT is implemented as follows. Consider a database with i directory attributes and D descriptors per attributes. Then, the CDT consists of iD entries. Each entry is formed with a descriptor id followed by a list of cluster definitions and their corresponding secondary memory addresses.

1. Directory Management for an Insert Request

We present two procedures below. The first one is executed during the descriptors search phase and the second one is executed during the address generation phase.

PROCEDURE DESCRIPTOR SEARCH

Purpose: Given a record for insertion, it determines the corresponding descriptor for each attribute-value pair in the record.

Input: The record to be inserted. Let there be n attribute-value pairs in the record for insertion. Thus, the record for insertion is $(<A_1,V_1>, <A_2,V_2>, \ldots, <A_n,V_n>)$.

Output: A set of k descriptors, where k is the number of directory attributes. These descriptors are output in the array D(1), D(2), ..., D(k).

Step 1: Set i = 1. Set j = 1.

Step 2: Consider the attribute-value pair $<A_i,V_i>$. Follow the pointer in PT, P_i, to the location in DDIT where the descriptors of A_i are stored. Let $m = P_i$. Set P_{i+1} to the next pointer in PT after P_i. If no pointer P_i, is found, go to Step 4.

Step 3: Check if $<A_i,V_i>$ can be derived from the m-th descriptor of DDIT. If so, set D(j) = m-th descriptor of DDIT, j = j+1, and go to Step 4. Else, go to step 5.

Step 4: i = i+1. If i > n, then go to Step 6. Else, go to Step 2.

Step 5: Set m = m+1. If $m < P_{i+1}$, go to Step 3. Else, set D(j) = null descriptor on A_i, j = j+1 and go to Step 4.

Step 6: Output D(1), D(2), ..., D(k). Terminate.

PROCEDURE ADDRESS GENERATE

Purpose: Generates the cluster to which the record for insertion belongs and its secondary memory address or addresses.

Input: D(1), D(2), ..., D(k) which was output from the previous algorithm.

Output: The cluster to which the record belongs and the secondary memory addresses, of this cluster.

Step 1: Consider the non-null descriptors in D(1), D(2), ..., D(k) and let these be ND_1, ND_2, ... ND_p. Let the attributes in the non-null descriptors be AT_1, AT_2, ..., AT_p. Let there be d_i descriptors in the database for attribute AT_i. Choose that attribute, AT_j, in AT_1, AT_2, ... AT_p with the maximum number of descriptors on it, d_j.

Step 2: Access the CDT entry for attribute AT_j and descriptor ND_j. Search this entry until the cluster defined by D(1), D(2),..., D(k) is encountered.

Step 3: Output the cluster number and its corresponding secondary addresses as stored in the entry. Terminate.

2. Directory Management for a Non-Insert Request

Once again, we shall present two procedures. The first one is executed by MDBS during the descriptors search phase and the second one is executed during the address generation phase.

PROCEDURE DESCRIPTOR SEARCH

Purpose: Given a non-insert request with a query, it determines the corresponding descriptor for each predicate in the query.

Input: The query in disjunctive normal form. Let there be i conjunctions and let there be P_j predicates in the j-th conjunction.

Output: A two-dimensional array D in which D(x,y) contains the corresponding dewcriptor for the y-th predicate of the x-th conjunction.

Step 1: Set m = 1. [At any point, the m-th conjunction is being examined. Set n = 1. [At any point, the n-th predicate of the m-th conjunction is being examined.] Set k = 1. Let the array D be initially empty.

Step 2: Let the attribute in the n-th predicate of the m-th conjunction be A and let its value be V. Use PT to determine the location in DDIT where the descriptors corresponding to A are stored. Set P and START to this value. Set END to the location in DDIT where the descriptors corresponding to A end.

Step 3: Check if <A,V> can be derived from the p-th descriptor in DDIT. If so, set D(m,n) to the p-th descriptor of DDIT and go to Step 4. Else, set p = p + 1. If p ≤ END, go to Step 3. Else, go to Step 4.

Step 4: Set n = n + 1. If n > P_m, then go to Step 5. Else, go to Step 2.

Step 5: Set m = m + 1. If m > i, then stop. Else, set n = 1 and go to Step 2.

PROCEDURE ADDRESS GENERATE

Purpose: Generates the clusters which will satisfy a user request and their secondary memory addresses.

Input: The two-dimensional array D from the previous procedure and the user query of i conjunctions.

Output: The clusters which will satisfy the user request and their secondary memory addresses.

Step 1: Set m = 1 [At any time, we examine the m-th conjunction]. Let query conjunction m have P_m predicates.

Step 2: Set n = P_m. Consider the descriptors D(m,1), D(m,2), ..., D(m,n). Let the attributes in these descriptors be AT_1, AT_2, ..., AT_n. Also, let D(m,i), $1 \leq i \leq n$, be the x_i-th descriptor on AT_i. Also, let there be d_i descriptors in all, in the database, on AT_i.

Step 3: Associate a count C_i with each AT_i as follows. If the i-th predicate of the m-th conjunction contains the operator '=', $C_i = 1/d_i$. If the i-th predicate of the m-th conjunction contains the operator '<' or '≤', then set $C_i = x_i/d_i$. Else, set $C_i = (d_i - x_i)/d_i$.

Step 4: Choose the AT_i with minimum C_i. Access the entry in the CDT for attribute AT_i and descriptor D(m,i). If the i-th predicate of the m-th conjunction is '<' or '≤', then also search the entries corresponding to attribute AT_i and all descriptors 'less than' D(m,i). Else, if the i-th predicate of the m-th conjunction

is '>' or '≥', then also search the entries corresponding to attribute AT_i and all descriptors 'greater than' D(m,i).

Step 5: For each cluster encountered in searching these entries, do Step 6.

Step 6: Set p = 1. Consider attribute AT_p and the p-th predicate of the i-th conjunction. If the predicate is '=', then check to see if the cluster definition includes descriptor D(m,p). If the predicate is '<', or '≤', then check to see if the cluster definition includes descriptor D(m,p) or any descriptor 'less than' D(m,p). Otherwise, check to see if the cluster definition includes descriptor D(m,p) or any descriptor 'greater than' D(m,p). If the checks fail at any point, then the cluster being examined does not satisfy the user request. Otherwise, set p = p + 1. If p > n, then output the cluster being examined and its corresponding address (or addresses). Else, go to Step 6.

Step 7: Set m = m + 1. If m > i, Terminate. Else, go to Step 2.

APPENDIX D: ALGORITHM FOR BINARY SEARCH OF DESCRIPTORS

In this appendix, we shall calculate the time for a binary search of N descriptors. Let

tpr :	time taken to perform an arithmetic operation
td :	time taken to read a descriptor
N :	number of descriptors
K :	descriptor being searched for
g(a) :	the nearest integer greater than or equal to a
h(a) :	the nearest integer less than or equal to a
lg :	logarithm of the base 2

Let the N descriptors we are searching be K_1, K_2, ... K_N. Then, the algorithm for binary search of these N descriptors is as follows.

1. $i = g(N/2)$, $m = g(N/2)$
2. If $K < K_i$ go to 3
   If $K > K_i$ go to 4
   If $K = K_i$ success
3. If $m = 0$ failure
   else $i = i - g(m/2)$
   $m = h(m/2)$
   go to 2
4. If $m = 0$ failure
   else $i = i + g(m/2)$
   $m = h(m/2)$
   go to 2

From the above, the total time to do binary search on N descriptors is

$$2tpr + lg(N(td + 4tpr))$$

APPENDIX E: I/O SUBMODEL FOR DISK SYSTEM

The I/O submodel consisting of M drive queues and one channel queue is shown in Figure 20. The inter-arrival distribution is chosen as exponential with L as the mean rate at which requests are received by the disk system. The inter-arrival distribution of requests to each drive queue is also exponential with a mean request rate of L/M. Let us analyze the drive queues and the channel queue, in turn.

Analyzing a Drive Queue - The drive queue service time is the seek time. The seek time of a disk drive is approximated by an equation of the form $(a + by)$, where a and b are constants and y is the number of cylinders traversed during the seek. We let N be the total number of cylinders in a disk drive. Then,

Mean service time = Mean seek time

$$= \sum_{y=1}^{N} (a + by)\left(\frac{2}{N} - \frac{2y}{N^2}\right)$$

$$= 2a - \frac{2a}{N^2}\sum_{y=1}^{N} y + \frac{2b}{N}\sum_{y=1}^{N} y - \frac{2b}{N^2}\sum_{y=1}^{N} y^2$$

$$= a - \frac{a}{N} + \frac{bN}{3} - \frac{b}{3N}$$

Second moment of service time

$$= \sum_{y=1}^{N} (a^2 + b^2y^2 + 2aby)\left(\frac{2}{N} - \frac{2y}{N^2}\right)$$

$$= a^2 - \frac{a^2}{N} + \frac{b^2N^2}{6} - \frac{b^2}{6} + \frac{2abN}{3} - \frac{2ab}{3N}$$

Variance of service time

$$= \text{Second moment of service time} - (\text{Mean service time})^2$$

$$= \frac{b^2N^2}{18} + \frac{b^2}{18} - \frac{a^2}{N^2} - \frac{b^2}{9N^2} - \frac{2ab}{3N^2} + \frac{a^2}{N} + \frac{2ab}{3}$$

Analyzing the Channel Queue - It is easy to see that

$$\text{Mean service time} = \text{disk rotation time} = d$$

$$\text{Variance of service time} = 0$$

$$\text{Mean Inter-arrival time} = \frac{1}{L}$$

$$\text{Variance of Inter-arrival time} = \frac{1}{L^2}$$

The channel queue is analyzed as an M/G/1 queue [Klei75] whose waiting time is

$$\frac{Ld^2}{2(1-Ld)}$$

Let total time in the channel system be the sum of the channel wait and service times. Let

$\bar{w}$ = channel wait time

$\bar{x}$ = channel service time

$\bar{s}$ = mean time in channel

$\overline{s^2}$ = second moment of time in channel

var(s) = $\overline{s^2} - (\bar{s})^2$ = variance of time in channel

Then,

$$\bar{s} = \overline{(w+x)} = \bar{w} + \bar{x}$$

$$= \frac{Ld^2}{2(1-Ld)} + d$$

$$\overline{s^2} = \overline{(w+x)^2} = \overline{w^2} + \overline{x^2} + 2\bar{x}\bar{w}$$

$$= \overline{w^2} + d^2 + \frac{Ld^3}{(1-Ld)} \quad \ldots\ldots\ldots\ldots (1)$$

$$\overline{w^2} = \frac{L^2d^4}{2(1-Ld)^2} + \frac{Ld^3}{3(1-Ld)}$$

Substituting $\overline{w^2}$ in (1),

$$\overline{s^2} = \frac{L^2d^4}{2(1-Ld)^2} + \frac{Ld^3}{3(1-Ld)} + d^2 + \frac{Ld^3}{(1-Ld)}$$

Finally,

$$\text{var}(s) = \overline{s^2} - (\bar{s})^2$$

$$= \frac{L^2d^4}{4(1-Ld)^2} + \frac{Ld^3}{3(1-Ld)}$$

APPENDIX F: ADDRESS GENERATION TIMES

The time taken for address generation is really the time taken to search the augmented CDT for the addresses of the corresponding set of clusters. We assume that the augmented CDT is physically implemented as follows. Consider a database with i directory attributes and D descriptors per attribute. Then, the augmented CDT consists of iD entries. Each entry is formed with a descriptor id followed by a list of cluster definitions and their corresponding secondary addresses. Let

i	:	number of directory attributes
D	:	number of descriptors per attribute
t	:	number of bytes needed to store a track address
p	:	number of bytes needed to indicate a back-end number
c	:	size of a cluster in tracks
lg	:	logarithm to base 2
u(x)	:	nearest integer greater than or equal to x
n	:	number of back-ends
s	:	track size
z	:	time to access a track from secondary memory
x	:	size of a CDT entry in the centralized strategy
adgen	:	time for address generation in the centralized strategy
y	:	size of a CDT entry in the other strategies
adgen1	:	time for address generation in the other strategies

The size, x, of each CDT entry in the centralized strategy is

$$u(\frac{lgD}{8}) + D^{i-1}(u(\frac{lgD}{8})i + c(t+p))$$

So,

$$adgen = \frac{xz}{s}$$

Similarly, the size y of each CDT entry in the remaining strategies is

$$u(\frac{lgD}{8}) + \frac{cD^{i-1}}{n}(u(\frac{lgD}{8})i + t), \text{ if } c \leq n;$$

$$u(\frac{lgD}{8}) + D^{i-1}(u(\frac{lgD}{8})i + \frac{ct}{n}), \text{ otherwise}$$

Finally,

$$adgen1 = \frac{yz}{s}$$

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93940	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93940	1
Chairman, Code 52Hq Department of Computer Science Naval Postgraduate School Monterey, CA 93940	25